21世纪高职高专规划教材

软件专业系列

案例式教材

软件测试

曹薇 主编

清华大学出版社



21世纪高职高专规划教材·软件专业系列

软 件 测 试

曹 薇 主编

清华大学出版社
北 京

内 容 简 介

本书较系统地介绍了软件测试的概念、原理、方法和技术。主要内容包括：软件测试的目标和原则，软件测试的认识误区，软件测试的定义和分类，测试过程管理，测试驱动开发的概念及策略，软件测试文档的分类和编制规范，黑盒和白盒测试用例的设计策略，单元测试的概念及策略，使用 CppUnit 进行单元测试，集成测试的概念及策略，系统测试的概念、分类及策略，面向对象软件测试的概念及策略，自动化测试的概念及主流测试工具，使用 WinRunner 进行功能测试，软件质量和质量保证；并在书末给出了软件测试案例。本书含有较多的例题、案例和习题，便于教学和自学。

本书强调理论与实践相结合，内容简明易懂、逻辑性强。可作为高职高专院校计算机专业的教材或参考书，也可供其他各类人员参考使用。

图书在版编目（CIP）数据

软件测试/曹薇主编. —北京：清华大学出版社，2008.3
21 世纪高职高专规划教材. 软件专业系列
ISBN 978-7-302-16966-6

Ⅰ. 软… Ⅱ. 曹… Ⅲ. 软件—测试—高等学校：技术学校—教材 Ⅳ. TP311.5

中国版本图书馆 CIP 数据核字(2008)第 013181 号

责任编辑： 束传政
责任校对： 李 梅
责任印制：

出版发行： 清华大学出版社　　**地　址：** 北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　编：** 100084
社 总 机： 010-62770175　　**邮　购：** 010-62786544
投稿与读者服务： 010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈： 010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　销： 全国新华书店
开　本： 185×260　**印　张：** 17.25　**字　数：** 393 千字
版　次： 2008 年 3 月第 1 版　**印　次：** 2008 年 3 月第 1 次印刷
印　数： 1～ 000
定　价： .00 元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。联系电话：010-62770177 转 3103　　产品编号：023497-01

前　言

随着信息技术的飞速发展，软件在社会生活的方方面面发挥着日益重要的作用。软件的复杂性不断增大，开发周期有限，这些都导致了软件质量时常得不到保证。如何提高软件开发的质量呢？除了按照软件工程的过程和方法论去优化需求分析、设计、编程外，通过软件测试发现软件中的缺陷并进行修复显然是提升软件质量的重要途径。

软件测试作为软件开发中的一个重要环节，已逐渐形成一门新的学科和产业。2003 年 10 月，国家人事部和信息产业部联合发文，将"软件评测师"资格考试纳入计算机技术与软件专业技术资格(水平)考试，就是一个显著的标志。

我国的软件测试研究和应用起步较晚，在软件测试理论研究、软件测试工具及框架研发、软件测试过程管理、软件质量保证等方面还落后于一些发达国家。尤其是在软件测试工具方面，国外的产品几乎占据了垄断地位。同时，软件测试的新理论与新技术日新月异。这些对我国的软件从业人员和测试从业人员都是严峻的考验。所以，作为软件测试产业从业人员的后备力量——广大的高职高专院校计算机专业学生(尤其是软件专业学生)，理应系统地学习软件测试的概念、原理、方法和技术，为将来投身我国的测试产业做好准备，力争为我国测试产业的发展做出应有的贡献。

本书结合高职高专院校的教学特点，较系统地对软件测试进行了介绍，力求逻辑严谨、简明易懂、实践性强。书中的例题、案例和习题较多，便于教学和自学。本书还吸纳了软件测试中若干新理论和新技术，以反映软件测试的最新进展。

本书共分 11 章。第 1 章主要介绍了软件测试的目标，软件测试的认识误区，软件测试的发展史、现状及发展趋势；第 2 章介绍了软件测试的定义、对象及分类，V&V，软件测试过程模型和过程管理理念，测试驱动开发的概念和策略，软件测试的原则，软件测试文档；第 3～4 章介绍了黑盒和白盒测试用例的设计方法和策略；第 5 章介绍了单元测试的概念、步骤、环境和内容，单元测试用例的设计思路，用 CppUnit 进行单元测试；第 6～7 章介绍了集成测试、确认测试、系统测试和验收测试的概念和策略；第 8 章介绍了目前研究和应用的热点——面向对象软件测试的概念、模型和策略；第 9 章介绍了软件测试自动化的必要性和优点，自动化测试的认识误区，自动化测试的实施流程，自动化测试的原理和方法，主流测试工具，IBM Rational 和 HP Mercury 公司的自动化测试解决方案，使用 WinRunner 进行功能测试；第 10 章介绍了软件质量和质量模型，软件度量，CMM 及 CMMI，PSP 及 TSP，SQA；第 11 章给出了几个具体的测试案例。书末附有常用测试术语

表和测试站点推荐。

本书由曹薇任主编，张乃洲任副主编。张乃洲编写了第2章和第5章，其他章由曹薇编写。由于水平有限、时间仓促，书中难免存在不足之处，恳请专家和读者批评指正。

编　者

2007年12月

目　录

第 1 章

软件测试概述

本章要点：

- 软件缺陷及其产生原因。
- 软件测试的目标。
- 软件测试中的认识误区。
- 软件测试的发展史。
- 软件测试的国内外现状。
- 软件测试的发展趋势。

软件是与计算机系统操作相关的程序、数据和文档，是人类社会高度发展的产物，是人类智慧的结晶。从 20 世纪 50 年代初，软件技术不断取得进展，使软件在内涵、规模、开发方法、应用领域等方面都发生着日新月异的变化。软件越来越多地影响和改变人类生活的各方面。然而，软件构成及开发的日益复杂、软件应用领域的日益拓宽也使得人们常常受到有缺陷的软件的影响，软件缺陷给人们带来了许多物质上和精神上的损失。软件质量不断受到人们的重视，为了发现软件中的缺陷，保证软件质量，软件测试应运而生。本章主要介绍软件测试的目标、发展史、现状及发展趋势，以及人们对软件测试的认识误区，为后续章节的学习打下基础。

1.1　软件测试的意义

1.1.1　软件缺陷的典型例子

俗话说，人无完人，金无足赤。人类智慧的结晶——软件也难以尽善尽美，存在缺陷的事情常有发生，可将软件中的所有质量问题都称为软件缺陷。软件缺陷可小可大，然而软件的性质决定了即使是一个很小的缺陷，也很有可能给使用者带来极大的损失。下面就是一些造成较大影响的软件缺陷的例子。

1. 千年虫问题

20 世纪 70 年代，当时计算机的存储空间很小，程序员为了缩减程序所占的存储空间，将表示年份的 4 位数中的前两位去掉，如“1985”被表示为“85”。

这些程序员当然知道到了2000年这种表示方法会带来麻烦，然而，他们认为不会等到2000年，软件系统就应该更新换代了，所以并没有顾及未来的问题。

不幸的是，这些程序员采用的年份表达方法在2000年快要到来时仍被广泛使用，这时千年虫问题才引起世界各国众多行业尤其是银行业、零售业、电信业的高度重视。为解决千年虫问题，避免出现如2001年与1901年在计算机日期表达中的混淆不清，全世界已付出数千亿美元的代价。

2. 爱国者导弹中的软件缺陷

在第一次海湾战争中，美军最大的一次伤亡是1991年2月25日在沙特阿拉伯的宰赫兰兵营被伊拉克的飞毛腿导弹击中，死28人，伤98人。其原因是，大名鼎鼎的爱国者导弹的软件中存在缺陷，导弹运行100小时共形成343.3毫秒的积累误差，导致687米的距离偏差，因此未能成功地拦截飞毛腿导弹。

有意思的是，发现这一微小偏差的并不是爱国者导弹的设计者——美国人，而是思维严密的以色列人。

3. 迪斯尼的圣诞节礼物

1994年圣诞节前夕，迪斯尼公司发布了第一个面向儿童的多媒体光盘游戏—“狮子王童话”。这是迪斯尼公司第一次进军儿童计算机游戏市场，由于该公司的品牌效应以及大力的广告宣传，“狮子王童话”的市场销售情况非常好，该游戏成为大多数父母圣诞节为孩子必买的礼物。

但随后的情况却出人意料。12月26日，很多客户反映该游戏在自己的机器上无法成功安装或无法正常使用。后来才证实，出现这种情况的原因是迪斯尼公司没有针对“狮子王童话”可能使用的各种机型进行系统兼容性测试，只是对少数机型进行了兼容性测试，所以导致该款游戏只能在少数几种机器上成功安装和运行。

4. 阿丽亚娜火箭发射失败

1996年6月，欧洲阿丽亚娜5型火箭第一次发射，由于定位软件出错，导致计算机命令固态推进器与主发动机尾喷管发生偏离，结果火箭发射升空仅40秒就爆炸了。

5. “冲击波”计算机病毒

2003年8月，“冲击波”计算机病毒首先在美国发作，导致美国政府机关、企业和个人成千上万台计算机受到攻击。随后，“冲击波”蠕虫病毒很快在因特网上广泛传播，中国、日本、欧洲等地的用户也受到了攻击，结果是大量的邮件服务器瘫痪，给整个世界范围内的网络通信带来了惨重的损失。

制造“冲击波”病毒的黑客只用了3周时间就完成了该病毒程序。该病毒仅仅利用微软公司Messenger Service中的一个缺陷，就攻破了计算机安全屏障，使所有基于Windows操作系统的计算机崩溃。更令计算机安全专家担忧的是，如不立即采用有效的防御措施，黑客将很快找到利用该缺陷控制大部分计算机的方法。

随后，微软公司紧急发布了升级补丁，以修复操作系统中存在的缺陷，抵御该病毒的攻击。

6. 微软 64 位服务器软件缺陷

2004 年上半年，微软公司承认，如果客户使用的是 64 位的 Windows Server 2003 企业版，并且硬件配置是英特尔安腾芯片，很可能突然死机。更可怕的情况是，死机后根本不可能重新启动，将给企业带来巨大的损失。

微软公司称，该问题是由硬件管理程序在检测硬件设备时造成的，随后即推出了升级补丁程序。

7. 索尼电视软件缺陷

2006 年 2 月，索尼（中国）公司称，2005 年下半年在中国内地推出的 5 款电视，包括液晶电视和液晶背投电视，由于在软件方面出现了设计缺陷，导致不能正常开关机。

索尼公司的专业人员研究后发现，特定范围内的液晶背投电视和液晶电视的软件中存在一个计时错误，该错误会导致相关型号电视在待机及累计工作约 1200 小时后，出现在使用中不能正常关机或在待机状态下不能开机的现象。而液晶电视的正常工作时间为 5 万小时。随后索尼（中国）有限公司对存在问题的 5 款电视进行了免费软件升级。

1.1.2　软件缺陷的产生原因

软件缺陷给人类带来的损失，是刚才的几个例子远远不能概括的。这种损失有经济上的、精神上的、身体上的，更有许多人因为软件缺陷而付出了生命的代价。

软件缺陷产生的原因到底是什么呢？软件产品是整个软件开发活动的成果结晶，所以我们应该到软件开发活动的各阶段去寻找原因。

按照软件工程的观点，在对软件系统的可行性进行论证之后，软件开发的主要阶段依次为需求分析、软件设计、软件编码、软件测试、软件运行和维护，如图 1-1 所示。据众多软件从业人员的亲身项目实践，得出的结论是，软件需求分析不够全面、准确是导致软件缺陷的最主要原因。

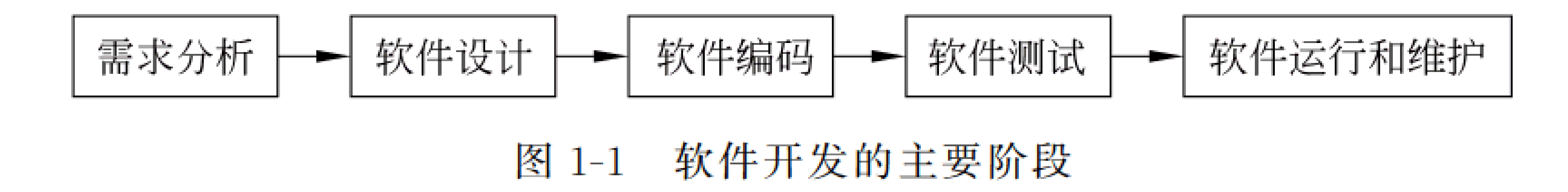


图 1-1　软件开发的主要阶段

需求分析的主要任务是确定待开发软件的功能需求、性能需求及运行环境约束。简单地说，需求分析是确定待开发软件“做什么”。由于软件开发最终是要交付用户使用的，因此所谓“需求”是指用户对软件的需求。系统分析人员和开发人员应在与用户反复、充分沟通的基础上完成需求分析。

由于软件是复杂的逻辑产品，用户在最终拿到软件之前往往很难一次性地精确描述对软件的需求，再加上系统分析人员、开发人员、用户对软件需求的关注角度和描述方式的不同，使得对需求的全面、准确的理解往往不能在需求分析阶段一蹴而就，而是随着软件开发活动的进行不断得到深化。需求分析阶段确定的需求不全面、不准确，即为软件缺陷的产生埋下了祸根。

软件设计和编码过程中的失误也会导致软件缺陷的产生，例如软件设计阶段考虑问题的片面性，软件设计文档不够具体，编码阶段的错误等。但很多情况下，不正确的软件

设计是不正确的需求分析引起的，编码阶段出现的错误则是由需求分析和软件设计不够完善、准确引起的。

1.1.3 软件测试的目标

尽管软件缺陷产生的原因已为很多人所知，但由于软件本身以及软件开发活动的特点，软件缺陷很难根除。所以，在软件交付使用前，为了尽量消除软件中的缺陷，对其进行测试是必不可少的。

软件测试的目标在早期被认为是尽可能多地发现软件中的潜在错误，这一点也可以从 Glenford J. Myers 给出的以下关于软件测试的规则描述中看出。

- 测试是为了发现程序中的错误而执行程序的过程。
- 好的测试方案是尽可能发现迄今为止尚未发现的错误的测试方案。
- 成功的测试是发现了迄今为止尚未发现的错误的测试。

当前仍然有部分人对软件测试存在误解，他们认为软件测试就是要证明软件是正确、可用的，能够满足用户的需要，而不是尽可能多地暴露软件中的潜在错误。

首先，这种想法是行不通的。由于软件是一种复杂的逻辑产品，对软件进行穷举测试是不可能的。因此，即使到目前为止对一个软件的测试中未发现任何错误，也不能说明该软件是绝对正确的，正所谓“软件测试只能证明软件有错，不能证明软件无错”。

从另一个角度说，对软件测试存有这种误解的人在进行测试时，往往在心理上会忽略软件中可能存在的缺陷，而把注意力集中在软件能否完成基本的、已知的功能上，这样的测试显然不是成功的测试。

1983 年，在 Glenford J. Myers 观点的基础上，Bill Hetzel 指出，软件测试的目标不仅是尽可能多地发现软件中的错误，还要对软件质量进行度量和评估，以提高软件质量。这一论断将对软件测试的认识提升到更高的层次。

1983 年，IEEE 对软件测试的定义则指出，软件测试的目标是为了检验软件系统是否满足用户的需求。

1.2 软件测试中的认识误区

软件测试的概念处于不断的发展之中，再加上缺乏软件测试的理论知识和实践经验，使得一些人对软件测试的认识存在误区。以下列举对软件测试的一些典型认识误区。

误区 1 测试和调试是一回事。

测试和调试有着根本的不同。测试是一个有计划、可重复的过程，目的是为了发现软件中的潜在错误和缺陷；而调试是一个随机的、不可重复的过程，目的是寻找错误的原因和具体位置，并修复错误。调试一般在测试后进行，当然，调试之后很可能又要进行测试，所以两者常交叉进行。

误区 2 可以对软件进行穷举测试。

对软件进行穷举测试是不可能的。也就是说，不可能对软件进行完全的测试以发现软件中的所有错误和缺陷。

这主要是因为，由于需求规格说明的复杂性和程序逻辑的复杂性，如下的测试是难以

做到的。

- 测试程序中的所有输入条件的取值。
- 测试程序中所有输入条件取值的组合。
- 测试程序中的所有路径。
- 测试出程序中所有潜在的错误和缺陷，例如由于需求分析不完善而导致的错误。

误区 3　若交付使用的软件有缺陷，是测试人员的失职。

软件测试的主要目标是发现软件中更多的错误和缺陷，但不能通过已有的测试证明某一个软件是绝对正确的。

交付使用的软件有缺陷，与多方面的人员都有关系，如系统分析人员、设计人员、编码人员、测试人员。软件开发中的任何一个环节出现问题，都有可能使软件出现缺陷。当然，这并不是说软件测试人员可以推卸责任，开发团队中多方面的人员加强沟通、合作才是最重要的。

误区 4　关注测试的执行而忽略测试用例的设计。

测试用例（Test Case）是为特定目标开发的一组测试输入、执行条件和预期结果，其目的是测试程序中的某路径，或核实程序或软件是否满足某个特定的需求。

从一定意义上说，设计测试用例是软件测试活动中具有核心地位的一个环节。若不注重测试用例的设计，很可能会遗漏有价值的测试用例，或导致设计出的测试用例不够准确，从而不可能对软件进行充分、有效的测试。

误区 5　测试比编程容易得多。

从某种意义上讲，对软件测试人员的要求比编程人员的要求高。这是因为，测试是一件十分复杂的工作。测试人员应具有细致沉稳的性格，很强的专业素质，对被测试软件的功能及架构十分清楚；测试人员还要能在无法实现穷举测试的前提下编写若干有价值的测试用例，以尽可能地揭露软件中的错误和缺陷；在进行自动化测试时，测试人员还应有编写测试脚本的能力。

误区 6　测试是编码之后进行的工作。

从软件开发的瀑布模型，人们容易得出这一错误的结论。事实上，软件生命周期中的测试阶段只是表明该阶段的主要工作是测试，并不意味着测试工作不能在需求分析阶段、设计阶段和编码阶段进行。

软件测试不仅是对程序的测试，需求分析和设计也应成为测试的对象。大量测试实践表明，软件中的大部分错误是在编码之前造成的，需求分析和设计造成的软件错误约占所有错误的 63%，而编码造成的错误仅占 37%。软件中的错误被发现得越晚，为修复它所付出的代价就越大。

所以在软件开发过程中，应尽早地、全面地开展测试。软件测试应成为一个独立的流程，可贯穿到软件开发的其他各流程，如需求分析、设计、编码，并与之并发地执行。在某流程中，达到恰当的测试就绪点即可进行独立的测试工作。测试还应是可迭代的。只有做到这些，测试才可能是成功的。

误区 7　测试自动化是万能的。

软件测试自动化可以提高测试的效率，但成本较高，需要自动化测试工具，还需要测

试人员编写测试脚本等。因此,只有需要经常执行的测试用例才适合于自动化测试。

当对某软件的测试自动化达到一定程度时,再想提高其自动化程度将会变得十分困难,需要付出很大的成本。2/8 原则同样适用于测试自动化,即付出 20%的成本可以实现 80%的测试自动化,若要实现剩余的 20%测试自动化,还需付出 80%的成本。所以在开发一个软件的过程中,不应盲目地提高其测试自动化的程度。

误区 8 软件测试是一种破坏性工作。

一些人认为,以尽可能发现软件中的错误为主要目标的软件测试是一种破坏性工作,是对软件开发人员工作的否定。这种想法在很大程度上制约了软件测试的发展,且十分不利于软件质量的保证和提高。

1.3 软件测试的发展史及现状

1.3.1 软件测试的发展史

软件测试是伴随着软件开发活动的产生而产生的。早在 20 世纪 50 年代,英国著名的计算机科学家图灵就给出了软件测试的原始定义。图灵认为,软件测试是程序正确性证明的一种极端的实验形式。这导致人们对软件测试的理解较为狭隘。

当时的软件测试主要针对用机器语言和汇编语言编写的程序,通过设计并运行测试用例,将运行结果与测试人员的预期结果进行比较,从而判断程序的正确性。人们在设计测试用例时常凭经验或直觉,这使得测试用例具有不完备性,测试也不够充分和有效,通过测试之后的软件往往还隐藏着大大小小的缺陷。

当时还有很多人认为,测试就是"调试",是查找软件中已发现错误的原因并纠正错误的一种活动。人们对软件测试不够重视,在测试上投入的人力、物力也很少,而且测试介入开发过程的时间点很晚,往往是在软件的编码结束后才进行测试。

直到 1957 年,人们才认识到测试应该是一种发现软件中潜在错误和缺陷的活动。1972 年,在北卡罗来纳大学举行了首届软件测试正式会议。1975 年,John Good Enough 和 Susan Gerhart 发表了名为《测试数据选择的原理》的文章,使软件测试得到了许多研究者的重视。1979 年,Glenford J. Myers 的著作《软件测试的艺术》(*The Art of Software Testing*)可以说是软件测试领域一本最重要的专著。Glenford J. Myers 在书中对软件测试的定义、目标等进行了描述,他认为测试是为发现错误而执行程序的过程,测试的目的在于尽可能多地发现软件中的错误。

20 世纪 80 年代早期,软件质量日益得到人们的重视。1983 年,Bill Hetzel 在《软件测试完全指南》(*Complete Guide of Software Testing*)一书中指出:"测试是以评价一个程序或系统属性为目标的任何一种活动,测试是对软件质量的度量。"

1983 年,IEEE 提出的软件工程术语指出:"测试是使用人工或自动的手段来运行或测定某个软件系统的过程,目的在于检验它是否满足规定的需求或弄清预期结果与实际结果之间的差别。"这个定义指出了软件测试的目的是检验软件系统是否满足需求。

进入 20 世纪 90 年代,软件测试得到了快速发展。随着面向对象分析和设计技术的普遍应用,面向对象的软件测试日益受到人们的重视,面向对象的软件测试的理论和技术

不断被完善;各种软件测试工具不断被开发出来,使软件测试的自动化程度不断提高;越来越多的测试模型被提出,测试的成熟度问题也得到众多专家的关注,并提出了若干测试成熟度模型。

1.3.2 软件测试的国内外现状

在软件业发达的国家,尤其是美国,软件测试得到了软件从业人员的普遍重视,已经形成了一个独立的产业,发展程度较高。主要体现在如下方面:

(1) 针对软件测试的理论研究在高校和科研院所如火如荼地进行,不断有软件测试方面的学术会议召开,理论研究成果层出不穷,引导着测试产业的发展方向;很多高校还单独设置了软件测试专业或开设了测试课程,大批学生投身于软件测试的理论学习和实践。

(2) 在软件开发企业尤其是比较成功的企业如微软、IBM 等,软件测试被放在十分重要的位置上,公司设有独立的测试部门,配备专职的测试人员。例如,在微软公司,开发工程师与测试工程师的比例是 1∶2,而国内的一般公司是 6∶1。在开发一个软件的过程中,花费在测试上的人力、物力和资金比花费在编程上的多得多。很多大型开发项目,测试会占据项目周期一半以上的时间。以 IE 4.0 为例,代码开发时间为 6 个月,而稳定程序花费了 8 个月的时间。这些都为公司开发出高质量的软件提供了有力的保障。

(3) 涌现出许多著名的软件测试工具提供商,如 HP Mercury Interactive(MI),Compuware,IBM Rational,Segue,Empirix 等。这些软件测试工具提供商的测试产品占据了全球的主流测试工具市场;同时,这些测试工具提供商所提供的测试方案和框架也极大地促进了软件测试技术的发展。

在我国,针对软件测试的理论研究起步于"六五"期间,在软件测试的实践方面则起步更晚。由于起步晚,我国无论在软件测试理论水平还是在测试产业方面,都与国际先进水平存在较大差距。但随着我国软件产业的高速发展及人们对软件质量的日益重视,软件测试正处于空前的快速发展时期,软件测试在我国正逐步形成一个独立的产业,主要体现在如下方面:

(1) 在我国,每两年召开一次的全国软件工程会议、全国容错计算会议都设有软件测试专题部分。从 2001 年起召开的全国测试学术会议,更是使软件测试得到了软件理论界前所未有的关注。

(2) 2003 年 10 月,国家人事部和信息产业部联合发文,决定将"软件评测师"资格考试纳入计算机技术与软件专业技术资格(水平)考试,使得软件测试人员的地位受到空前的重视。

(3) 国内著名的软件公司大都认识到软件测试的重要性,并成立了专职的软件测试队伍。虽然测试人员的数目和所占比例还不能与世界顶尖公司相比,但独立测试的意识在各公司得到了日益深化。

(4) "以测代评"正逐步成为我国科技项目择优支持的一项重要举措。如国家 863 计划对数据库管理系统、操作系统、办公软件、ERP 等项目的经费支持,都是通过第三方测试机构(即独立于软件开发商和用户的一方)科学、客观的测试结果来决定的。

(5) 各种软件测试职业培训机构如雨后春笋般涌现,软件测试还成为高校软件专业

普遍开设的课程，越来越多的人投身于软件测试的学习和实践中。

有理由相信，再经过若干年的发展，我国测试产业的发展水平会逐步接近国际先进水平，这也将促进我国软件产业的健康、快速发展。

但必须认识到，虽然近年来软件测试在国外和我国都取得了很大的进展，但其发展速度和水平仍难以适应高速发展的软件开发技术，软件测试从业人员面临着巨大的考验，主要体现在如下方面：

(1) 随着社会的发展，软件规模日益扩大，功能日益复杂，应用领域复杂多样，都对软件测试提出了新的考验。

(2) 面向对象开发是当前软件开发的主流技术，但面向对象软件的测试理论和技术还很不成熟。

(3) 对分布式系统的测试技术仍处于发展阶段。

(4) 对实时系统缺乏有效的测试手段。

(5) 对信息系统安全性的测试还处于起步阶段。

软件测试之所以发展得相对缓慢，主要有以下原因：

(1) 在软件测试的发展进程，尤其是软件测试的发展早期，软件开发人员对软件测试存在着敌对心理。认为软件测试一旦进行，就会找出程序中的许多错误，自己的劳动成果就会被部分否定，所以开发人员不愿意进行测试，更不愿意别人对自己的劳动成果进行测试。这种对软件测试的敌对心理，在很长一段时期内阻碍了软件测试的发展。

(2) 软件测试应贯穿软件开发的全过程，应把软件测试当作一个软件工程。也就是说，不能将软件测试当作是单一的几种方法或技能，软件测试也应遵循工程化原则，应当有完善的方法、过程和工具做支撑。但是，在软件概念出现以后的很多年，人们并没有软件工程的概念，这显然制约了软件测试的发展。20 世纪 70 年代以后，软件开发才进入软件工程阶段。软件工程的快速发展进一步加速了软件测试的发展。

(3) 软件测试是一门理论与实践高度结合的学科。没有系统的理论做支撑，软件测试就缺乏长足发展的后劲；而离开了测试实践，测试理论只能是纸上谈兵。但从现实情况来看，很少有人是既从事测试理论研究又参与大量测试实践的，这使得软件测试的发展面临着不平衡。

1.4 软件测试的发展趋势

纵观国内外软件测试现状，软件测试的主要发展趋势如下：

(1) 软件开发过程中的每个阶段都是可测试的，软件测试及早地介入软件开发的各阶段，如需求分析阶段和设计阶段。

(2) 软件测试成为一个完全独立的流程。测试可以贯穿软件开发的其他流程，并与之并发地执行。在某流程中，只要某测试达到准备就绪点，就可以进行该测试，而且测试是可迭代的。

(3) 面向对象的软件测试理论和技术不断发展。

(4) 分布式系统、实时系统的测试理论和技术不断发展。

（5）软件测试人员的地位得到极大的提高。软件公司大都设置了独立的软件测试部门，配备有专职的软件测试人员，测试部门在公司各部门中有着重要的地位。软件测试不再被看作是一种破坏性工作，只有具备很强软件开发能力且性格良好的人才能胜任软件测试工作。

（6）第三方测试迅速发展。第三方测试机构是一个具有中介性质的服务机构，它能通过自身专业化的测试技术和测试团队为客户和软件开发商提供有价值的服务。由独立的第三方软件测试机构完成测试，能够客观、公正地评价被测试软件，对软件开发商和用户来说都是有利的。

1.5　小结

软件测试是伴随着软件的发展而不断发展的。由于软件本身的特点，以及软件开发活动的特点，软件缺陷从来就难以根除，它可能给使用者造成或大或小的损失。

软件测试的目标是尽可能多地发现软件中的错误和缺陷，对软件质量进行度量和评估以提高软件质量，并检验软件系统是否满足用户需求。

对软件测试的看法中存在许多误区，只有认识这些误区才能更好地理解和实施软件测试。目前，软件测试在国内外均得到了快速发展，但软件测试的发展水平还远不能适应高速发展的软件开发技术，很多方面有待进一步研究和发展。

习　题

1. 为何软件缺陷难以避免？试谈谈你的观点。
2. 软件测试的目标是什么？
3. 测试和调试有何不同？
4. 为什么对软件进行穷举测试是不可能的？
5. 在一个软件的开发过程中，测试自动化的程度越高越好吗？为什么？
6. 软件测试的发展落后于软件开发技术的发展水平，主要原因是什么？
7. 什么是第三方测试，其有何优点？
8. 你认为软件测试的发展趋势是什么？

第 2 章

软件测试基础

本章要点：

- 软件测试的定义。
- 软件测试的对象。
- 验证与确认的含义和区别。
- 软件测试的分类。
- 软件测试过程模型。
- 测试驱动开发的思想。
- 软件测试的原则。
- 软件测试文档的作用和分类。

通过第 1 章的学习，我们对软件测试有了一个大致的认识，但对其基本概念，如软件测试的定义、对象、分类等并不十分清楚，本章的目的在于介绍关于软件测试的基本概念、原则及软件测试文档，以及目前流行的测试驱动开发方法。这些内容是学习后续章节的基础。

2.1 软件测试概念

2.1.1 软件测试的定义和对象

1. 软件测试定义

1979 年，Glenford J. Myers 在其著作《软件测试的艺术》(*The Art of Software Testing*)中，对软件测试定义为："测试是为了发现错误而执行的一个程序或系统的过程。"这个定义不管是在早期还是当今，都有着相当大的影响。

20 世纪 80 年代早期，软件质量越来越多地得到人们的重视，软件测试的内涵正悄悄发生着改变。测试不再单纯地被认为是一个发现错误的过程，还被作为软件质量保证(Software Quality Assurance，SQA)(软件质量保证将在第 10 章进行介绍)的主要职能，包含软件质量评价的内容。

1983 年，Bill Hetzel 在《软件测试完全指南》(*Complete Guide of Software Testing*)

一书中指出："测试是以评价一个程序或系统属性为目标的任何一种活动，测试是对软件质量的度量。"这个定义是对 Glenford J. Myers 定义的很好的补充，至今仍被引用。

1983 年，IEEE 提出的软件工程术语中对软件测试下的定义是："使用人工或自动的手段来运行或测定某个软件系统的过程，其目的在于检验它是否满足规定的需求或弄清预期结果与实际结果之间的差别。"这个定义明确地指出，软件测试的目的是检验软件系统是否满足需求。软件测试不再被认为是一个一次性的、只属于开发后期的活动，而应与软件的整个开发流程融为一体。

2. 软件测试的对象

早先许多人认为既然软件测试是希望发现程序中的错误，那么测试的对象就是程序。实际上，程序是软件开发过程中多个阶段工作的产物，如第 1 章所述，尽管软件中的错误会在程序中反映出来，但大部分错误是在编码之前造成的。

所以测试的对象不仅仅是程序，需求分析和设计工作也应列为测试的对象。甚至从一定意义上说，它们比程序更需要引起测试人员的注意。

2.1.2 验证与确认

目前，软件测试的效果难以尽如人意，是因为测试中尚存在如下问题：

(1) 测试工作多在软件开发的后期阶段进行，没有实现及早地、全面地进行测试，没有一个规范化、系统化的测试过程。

(2) 测试设计和测试操作未进行分离。

(3) 许多软件质量保证活动，如工作产品评估、可跟踪性分析、接口分析、关键性分析等是零散的、不自觉的行为，缺乏相应的规划。

为解决上述问题，人们提出了 V&V 概念，即验证(Verification)和确认(Validation)，这是软件测试领域十分有影响的概念。

1. 验证

验证即检验软件是否实现了预先定义的功能和其他特性，即判断软件开发每一阶段的活动是否已成功地完成，各开发阶段形成的软件配置是否保持一致。

2. 确认

确认也可理解为有效性确认。确认的目的在于判断交付使用的软件可否追溯到用户的需求。确认的作用是检验软件产品功能及其他特性的有效性。

3. 验证和确认的关系

从表面上看，对验证和确认的描述十分类似。它们的区别是什么？以下是 Boehm 对 V&V 两者的解释。

- Verification：Are we building the product right(我们在正确地构造软件吗)？
- Validation：Are we building the right product(我们在构造正确的软件吗)？

同样都是"正确"，但意义却不一样。举例来说，某软件的开发过程完全按照需求规格说明书和设计规格说明书的预期要求进行，每一开发阶段的工作都没有错误，且各阶段的软件配置具有一致性，显然此软件应该能通过验证，但一定能通过确认吗？不一定。因

为，也许该软件的需求分析和设计本身都存在缺陷，或者说开发人员（甚至也包括用户）原先对软件的功能及其实现的理解上存在缺陷，即使完全按预期的要求开发软件，也不一定能满足用户的最终真正需求。

所以，虽然定义中同样都含有“正确”，但确认定义中的“正确”的级别更高，它要求开发出来的软件对用户是真正有效的，能满足用户所有的最终需求，而这些需求中有些可能是潜在的，是用户先前都没有想到的。而验证定义中的“正确”只能说明软件开发的各阶段实现了既定的要求，但这些既定的要求本身可能存在问题。

验证和确认都属于测试活动。可以这样认为：

验证＋确认＝测试

验证和确认是不同级别的测试活动。

2.2 软件测试分类

下面从几个侧面介绍软件测试的分类。

2.2.1 按开发阶段分类

按开发阶段，软件测试可以划分为单元测试、集成测试、确认测试、系统测试和验收测试。

1. 单元测试

单元测试（Unit Testing）又称模块测试，是针对软件设计中的最小单位——程序模块进行正确性检验的测试。单元测试的目的在于发现程序模块内部可能存在的各种错误，检查各模块是否实现了详细设计说明中的模块功能、性能、接口及设计约束等方面的要求。单元测试应从程序模块的内部结构出发设计测试用例。多个模块可以并行地独立进行单元测试。

虽然单元测试是对某个模块的测试，但在测试某模块时，应考虑它与外界的联系。用一些辅助模块模拟与被测模块相联系的其他模块，从而达到测试被测模块的目的。

2. 集成测试

集成测试（Integrated Testing）也称为组装测试。在单元测试的基础上，将所有程序模块按照概要设计的要求组装成一个系统。集成测试的目的在于发现并排除在模块连接过程中可能出现的问题，最终构成符合概要设计要求的软件系统。

3. 确认测试

确认测试（Validation Testing）又称为有效性测试。确认测试的目的是检查已实现的软件系统是否满足需求规格说明书中规定的各种需求，以及软件配置是否完全、正确。

4. 系统测试

最终得到的软件不是一个孤立的对象，往往是作为整个目标系统的一个组成元素而存在的。

系统测试（System Testing）将通过确认测试的软件作为整个基于计算机系统的一个

元素，在实际运行环境下或模拟系统运行环境下，测试其与系统中其他元素（硬件、外设、网络、系统软件、支持平台等）能否正确地配置、连接，并满足用户需求。

系统测试的目的是通过与系统的需求定义比较，发现软件与系统定义不符合的地方。

5. 验收测试

验收测试（Acceptance Testing）即按项目任务书或合同、供需双方约定的验收依据文档对整个系统进行测试与评审，以决定是否接收软件系统。

验收测试是以用户为主的测试，但软件开发人员和 SQA（即 SQA 人员）也应参加。

2.2.2 按测试实施组织分类

按照实施测试的组织，可将测试分为 α 测试、β 测试和第三方测试。

1. α 测试

α 测试（Alpha Testing）属于开发方进行的测试，指软件开发方组织公司内部人员模拟各类用户对即将交付的软件产品（称为 α 版本）进行测试，以发现其中的错误并改正。α 测试的关键在于尽可能逼真地模拟软件的实际运行环境，并尽最大努力涵盖所有可能的用户操作方式。

α 测试的目的是评价软件产品的 FLURPS（Function，Localization，Usability，Reliability，Performance，Support），即功能、局域化、可使用性、可靠性、性能和支持，尤其注重产品的界面和特色。经过 α 测试调整的软件产品称为 β 版本。

2. β 测试

β 测试（Beta Testing）是用户进行的测试，但通常不等同于验收测试，即决定是否接收软件并不是 β 测试的目的。β 测试的目的在于帮助开发方在正式发布软件产品前对其进行最后的改进。

β 测试一般在 α 测试之后进行，是由大量用户在实际操作环境下对软件的 β 版本进行的测试。这些用户是与公司签订了支持软件产品预发行合同的外部客户，他们被要求以尽可能多的方式使用该软件，并将使用过程中出现的错误信息（真实的以及主观认定的）返回给开发方。开发方根据用户的错误报告，在正式发布软件产品之前对之进行一系列改进。

β 测试主要在于衡量产品的 FLURPS，着重于产品的支持性，包括文档、客户培训和支持产品生产能力。

3. 第三方测试

第三方测试是指由不同于开发方和用户方的组织进行的测试。通常模拟用户的真实操作环境，对软件进行确认测试。第三方测试有利于客观、公正地测试和评价软件。

2.2.3 按测试策略分类

根据测试实施策略的不同，软件测试可分为白盒测试、黑盒测试和灰盒测试。

1. 白盒测试

白盒测试（White-box Testing）又称为结构测试或逻辑驱动测试。顾名思义，“白盒”可理解为程序装在一个透明的盒子里，所以盒子内的程序对测试人员是可见的。

执行白盒测试的人员清楚地了解程序内部逻辑结构和处理过程，检查程序内部结构和路径是否达到了预期的设计要求。白盒测试方法将在第4章具体介绍。

2. 黑盒测试

黑盒测试(Black-box Testing)又称为功能测试或数据驱动测试。顾名思义，"黑盒"可理解为程序装在一个漆黑的盒子里，所以盒子内的程序对测试人员是不可见的。

执行黑盒测试的人员在已知软件应具有的功能的基础上，完全不考虑程序的内部逻辑结构和处理过程，在程序接口处进行测试，检查在需求规格说明书中规定的预期功能是否能正常实现。黑盒测试方法将在第3章具体介绍。

3. 灰盒测试

灰盒测试(Gray-box Testing)是一种介于白盒测试和黑盒测试之间的测试(故很多资料未对其单独介绍)。它基于程序运行的外部表现同时又结合程序内部逻辑结构来设计测试用例，执行程序并采集程序路径执行信息和外部用户接口结果。一般认为，集成测试阶段采用的测试策略近似于灰盒测试。

2.2.4 按测试执行方式分类

根据软件测试的执行方式，可将软件测试分为静态测试(Static Testing)和动态测试(Dynamic Testing)两种。

静态测试不实际执行程序，而是利用人工手段及静态测试工具完成对程序的静态测试。静态测试的主要目的是检查软件的表示与描述是否一致，没有冲突和歧义。静态测试将在第4章进行介绍。

动态测试则实际运行测试用例，以发现软件中的错误。

依据黑盒方法设计的测试是动态测试，白盒方法设计的测试则包括静态测试和动态测试两种类型，故对动态测试的介绍将贯穿于第3章和第4章。

2.2.5 其他测试方法和技术

在实际应用中，还有许多具体的测试类型，它们往往是为实现某特定目标而进行的测试。例如回归测试、迭代测试、功能测试、性能测试、安全性测试、可靠性测试、兼容性测试、可移植性测试、冒烟测试、用户界面测试、随机测试、引导测试、本地化测试等。

1. 回归测试

回归测试(Regression Testing)是为了验证对软件引入的修改的正确性及其影响而进行的测试。

在软件开发的任何一个阶段，只要软件发生了改变，就有可能引起一系列问题。软件的改变可能是因为发现了软件中的错误并做出了修改，也有可能是因为在集成或维护阶段加入了新的功能模块。

当软件中的错误被发现，往往需要修改的地方会有多处，包括表面的和深层次的。为了不遗漏任何需要修改之处，也为了避免修改对软件中未被修改之处造成的副作用，应进行回归测试；加入新的功能模块到软件中后，为使新功能能正常实现且不会对其他模块造成副作用，也应进行回归测试。

回归测试的工作量在整个软件测试过程中占有很大的比重，软件开发的各个阶段都会进行多次回归测试。

关于回归测试将在第 7 章详细介绍。

2. 迭代测试

迭代测试是从迭代的开发模式中延伸出来的。在迭代开发模式中，系统的开发是通过多次迭代完成的。每次迭代都包含需求分析、设计、编码、集成、测试等活动。每次迭代完成后，会出现新的迭代周期。通过一次次地迭代，系统增量式地集成若干新功能，直至最终实现系统应具备的全部功能。

在每个迭代周期中，测试工作由两方面组成：

(1) 对当前迭代周期产品的增量测试。

(2) 对原先迭代周期已完成功能的回归测试。

迭代开发模式继承了瀑布开发模式的优点：全面、计划性强和易于管理。更为重要的是，迭代开发模式将测试工作分布到每个迭代周期中，使测试工作提前进行，以尽早地发现软件中的缺陷，从而降低软件开发的风险和成本。

3. 功能测试

功能测试(Functional Testing)也称为行为测试(Behavioral Testing)，它根据产品特征、操作描述和用户方案，测试一个产品的特性和可操作行为以确定它们是否满足设计需求。一般把黑盒测试称为功能测试，当然这不是绝对的。功能测试有时也会用到白盒测试的方法，而黑盒测试有时可能是在对性能进行测试。

4. 性能测试

性能测试(Performance Testing)是评价一个产品或组件与性能需求是否符合的测试，包括负载(压力)测试、强度测试、容量测试、疲劳测试等类型。

性能测试是软件测试中的一个重点。对性能测试将在第 7 章进行具体介绍。

5. 安全性测试

安全性测试(Security Testing)的目的在于检测软件系统对非法侵入的防范能力。理论上讲，只要拥有足够多的时间和资源，任何系统都是可以侵入的，所以通过安全性测试及采取相应的措施后，应使非法侵入软件系统的代价大于被保护信息的价值，使非法侵入者得不偿失。

6. 可靠性测试

可靠性测试(Reliability Testing)的目的是测算在一定的环境下，系统能正常工作的概率。通常，用平均无故障时间(Mean Time Between Failures，MTBF)即两次失效之间的平均操作时间来衡量系统的可靠性。

7. 兼容性测试(配置测试)

兼容性测试(Compatibility Testing)有时也被称为配置测试(Configuration Testing)，但两者含义略有不同。一般说来，配置测试是为了保证软件在其相关的硬件上能够正常运行，而兼容性测试则主要是测试软件能否与其他不同的软件协作运行。

8. 可移植性测试

可移植性测试(Portability Testing)的目的在于验证软件能否被移植到指定的硬件或软件平台上。

9. 冒烟测试

冒烟测试(Smoke Testing)的对象是每一个新编译的需要正式测试的软件版本,目的是确认软件基本功能正常,可以进行后续的正式测试工作。冒烟测试的执行者是版本编译人员。

冒烟测试不能由测试小组独立建立,它应该是通过联合的方式,至少是在与开发人员达成一致的情况下建立的。冒烟测试的目标是显示其稳定性,而不是发现错误。必须在系统测试环境中进行冒烟测试。

冒烟测试的名称可以理解为该种测试耗时短,仅用一袋烟的功夫足够了。也有人认为是将其形象地类比于新电路板的基本功能检查(任何新电路板焊好后,应先通电检查,如果存在设计缺陷,电路板可能会短路,即出现冒烟的现象)。

10. 用户界面测试

用户界面测试(User Interface Testing)的目的在于测试用户界面的风格是否满足客户要求,包括用户友好性、人性化、易操作性等测试。

11. 随机测试

随机测试(Ad Hoc Testing)是没有书面测试用例的测试,主要是依据测试人员的经验对软件进行功能和性能抽查。

随机测试是根据测试文档执行用例测试的重要补充手段,是保证测试覆盖完整性的有效方式。

12. 引导测试

引导测试(Pilot Testing)是指在软件开发中验证系统在真实硬件和客户基础上处理典型操作的能力。在软件外包测试中,引导测试通常是客户检查软件测试公司测试能力的一种形式,只有通过了客户特定的引导测试,软件测试公司才能接受客户真实软件项目的软件测试。

13. 本地化测试

本地化测试(Localization Testing)的对象是软件的本地化版本,测试目的在于测试特定目标区域设置的软件本地化质量。从测试方法上可以分为基本功能测试、安装/卸载测试、当地区域的软硬件兼容性测试等。本地化测试的内容主要是软件本地化后的界面布局和软件翻译的语言质量,包含软件、文档和联机帮助等部分。

2.3 软件测试过程

随着测试技术的不断发展,人们更加关注测试的过程,对测试过程的管理已成为成功实施测试的重要保证。

软件测试过程用于定义软件测试的流程和方法。众所周知，开发过程的质量决定了软件的质量，同样地，测试过程的质量将直接影响测试实施的效果。软件测试过程和软件开发过程一样，都应遵循软件工程的原理。

2.3.1 软件测试过程模型

随着测试过程管理的发展，测试人员通过大量的实践总结出了很多很好的测试过程模型，如 V 模型、W 模型、H 模型等。这些模型将测试活动进行了抽象，并与开发活动进行了有机的结合，是测试过程管理的重要参考依据。

1. V 模型

V 模型最早是由 Paul Rook 在 20 世纪 80 年代后期提出的，旨在改进软件开发的效率和效果。V 模型反映出了测试活动与分析设计活动的关系，如图 2-1 所示。图中，从左到右描述了基本的开发过程和测试行为，非常明确地标注了测试过程中存在的不同类型的测试，并且清楚地描述了这些测试阶段和开发过程中各阶段的对应关系。

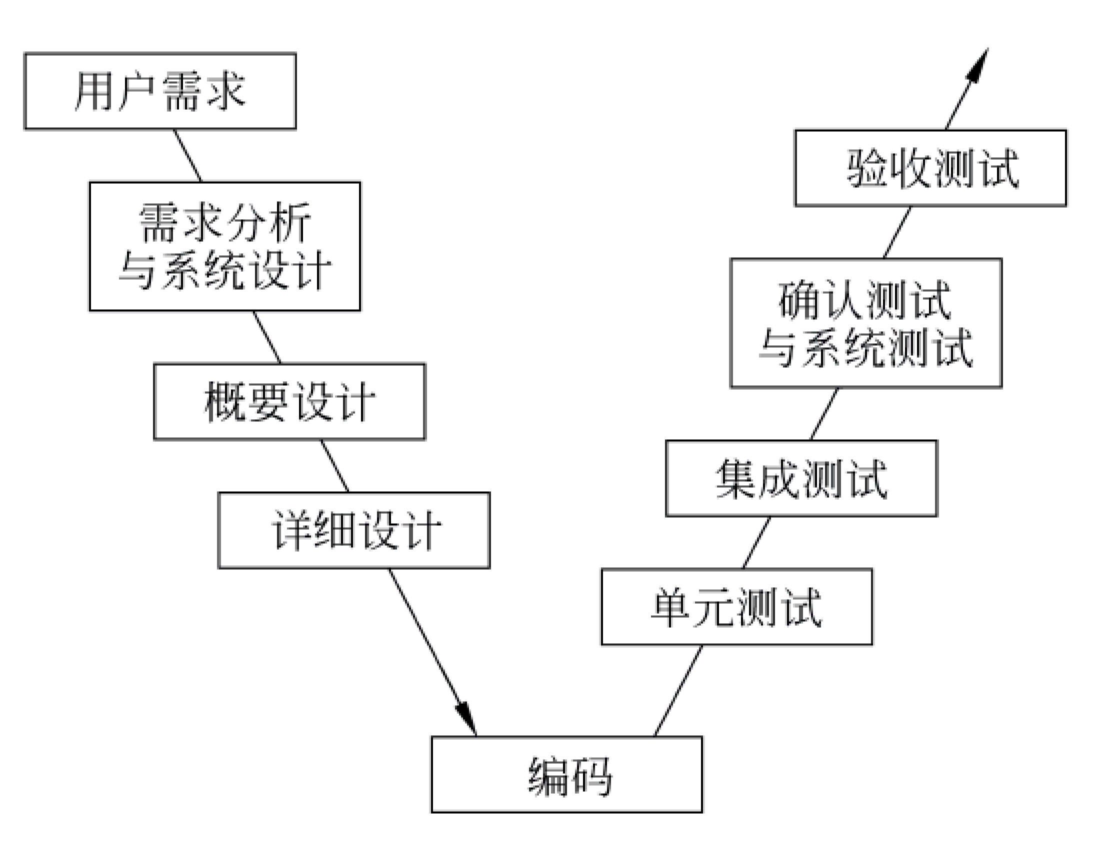


图 2-1 软件测试 V 模型

V 模型指出，单元和集成测试应检测程序的执行是否满足软件设计的要求；系统测试应检测系统功能、性能的质量特性是否达到系统要求的指标；确认测试和验收测试追溯软件需求规格说明书进行测试，确定软件的实现是否满足用户需要或合同的要求。

V 模型也存在一定的局限性。它仅仅把测试作为在编码之后的一个阶段，主要是针对程序进行的寻找错误的活动，对软件设计、需求分析等活动的测试要到后期才能完成。这样的测试顺序会使修复错误的代价大大增加，不利于提高软件开发和测试的效率。

2. W 模型

V 模型未能体现出“尽早地、全面地进行软件测试”的原则，为了弥补 V 模型的不足，W 模型出现了。

W 模型是由 Evolutif 公司提出的。相对于 V 模型，W 模型增加了软件各开发阶段中应同步进行的验证和确认活动。如图 2-2 所示，W 模型由两个 V 字型模型组成，分别表示测试和开发过程。从图 2-2 可以明显看出测试与开发的并行关系，也就是说，测试与开发是紧密结合的。

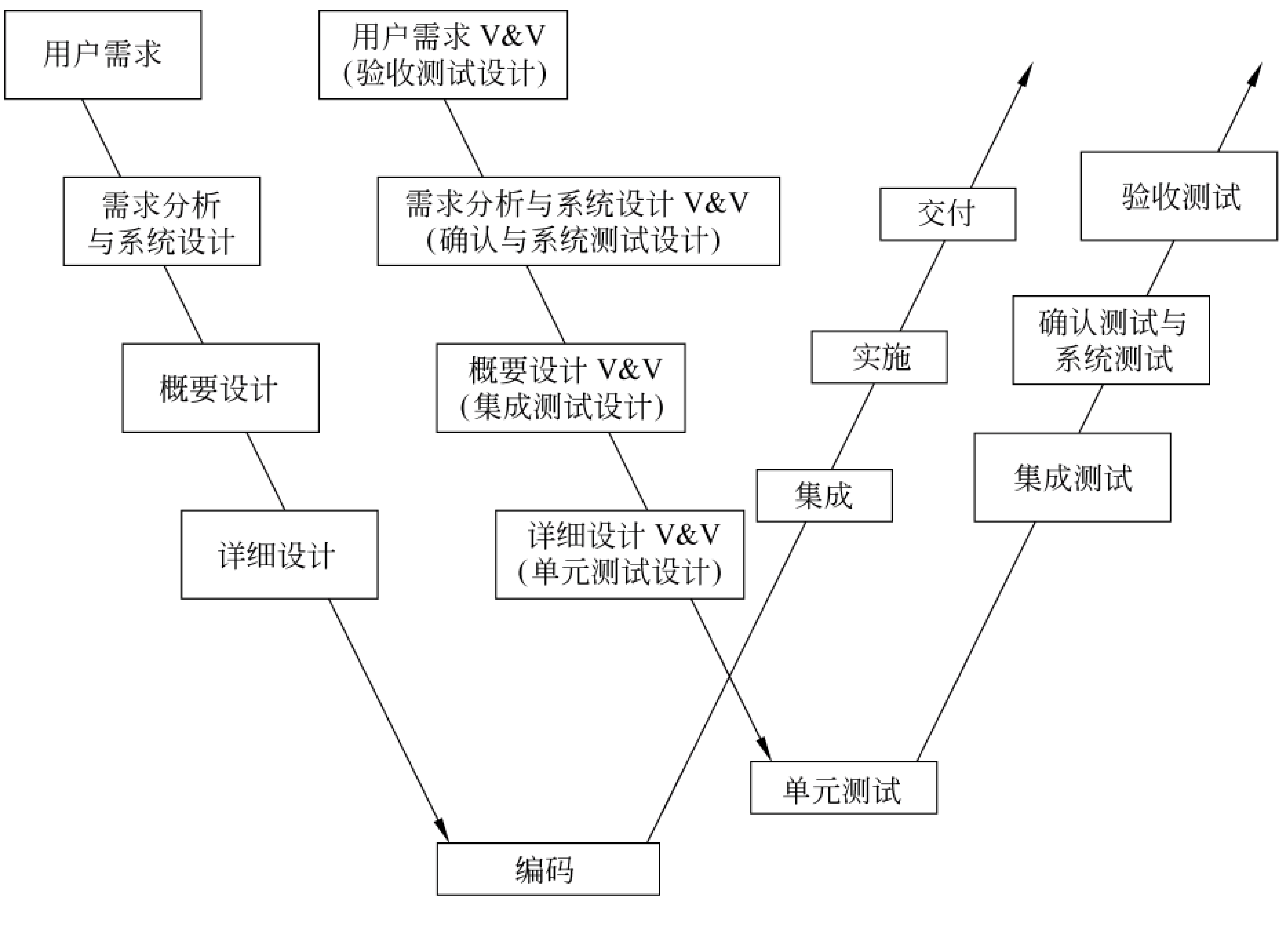


图 2-2 软件测试 W 模型

W 模型强调,测试伴随着软件开发的各阶段,测试的对象不仅仅是程序,需求分析、设计等同样需要测试。也就是说,测试与开发是同步进行的,当某一阶段的工作完成后,就可以进行测试。

W 模型有利于尽早地、全面地进行测试,以发现软件中存在的问题。例如,需求分析完成后,测试人员就应该参与到对需求的验证和确认活动中,以尽早地发现需求分析中存在的问题,并从可测试性角度为需求文档的编写提出建议。同时,测试人员结合前期对项目的把握,有利于及时了解项目的难度和测试中存在的风险,易于制定出完善的测试计划和方案,安排开发中各阶段的测试方法、进度和人员,使软件的开发过程进展顺利,提高软件测试和开发的效率。

W 模型也有利于全过程地测试。这是因为 W 模型中将测试与开发活动紧密结合起来,使测试人员充分关注开发过程,对开发过程的各种变更及时响应。例如,根据开发进度计划的变更及时调整测试进度和测试策略,以及依据需求的变更及时调整测试用例等。

W 模型也存在局限性。在 W 模型中,需求分析、设计、编码等活动被视为串行的,同时,测试和开发活动之间也是一种线性的关系,某开发活动完全结束后才可以正式开始进行测试,这样就无法支持迭代、自发性及变更调整。对于当前软件开发复杂多变的情况,W 模型并不能完全解决测试管理中面临的困惑。

3. H 模型

V 模型和 W 模型都存在不足之处,它们都把软件的开发过程中的需求分析、设计、编码等活动视为串行的。而大量的实践表明,各阶段保持严格的串行关系只是一种理想的状况,需求的变更等都会破坏这一理想状况,故与各开发阶段相对应的测试之间也不可能

保持严格的次序关系。同时，各层次的测试（单元测试、集成测试、系统测试等）也存在反复触发、迭代和增量关系。

为了解决以上问题，测试专家提出了 H 模型。它将测试活动完全独立出来，形成一个完全独立的流程，以将测试准备活动和测试执行活动清晰地体现出来，如图 2-3 所示。

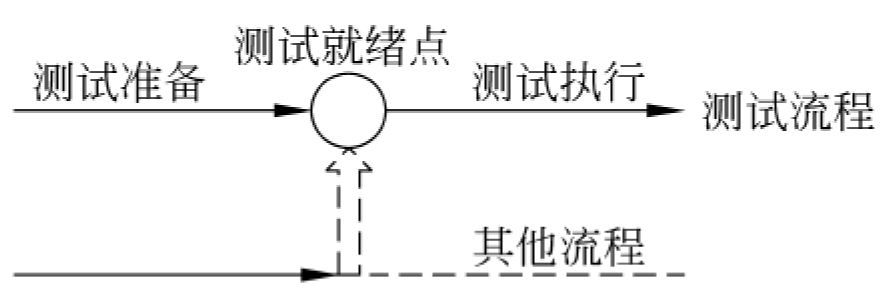


图 2-3　软件测试 H 模型

图 2-3 绘出的仅为整个软件生产周期中某个层次上的一次测试。图中标注的其他流程可以是任意的开发流程，例如设计流程或编码流程，也可以是非开发流程，如 SQA 流程，甚至是测试流程自身。只要测试准备活动完成，达到了测试就绪点，即可执行测试工作。

例如，在一个构件化 ERP 项目的系统测试过程中，由于前期需求难以确定，开发周期相对较长，为了进行更好的跟踪和管理，项目采用增量和迭代模型进行开发。整个项目开发共分三个阶段：第一阶段实现进销存的简单的功能和工作流；第二阶段实现固定资产管理、财务管理，并完善第一阶段的进销存功能；第三阶段增加办公自动化的管理。该项目每一阶段的工作是对上一阶段成果的一次迭代完善，同时叠加了新功能。

在该项目的系统测试过程中，根据 H 模型的思想，把系统测试作为一个独立的流程，达到相应的测试就绪点时即可执行测试。该系统的三个阶段相对独立，每一阶段完成的阶段产品亦具有相对独立性，可以作为系统测试的测试就绪点。故在该系统开发过程中，可开展三个阶段的系统测试，每个阶段系统测试具有不同的侧重点，以实现与各阶段开发的紧密结合，尽早发现软件中的错误，降低错误修复的成本。软件开发与系统测试过程的关系如图 2-4 所示。

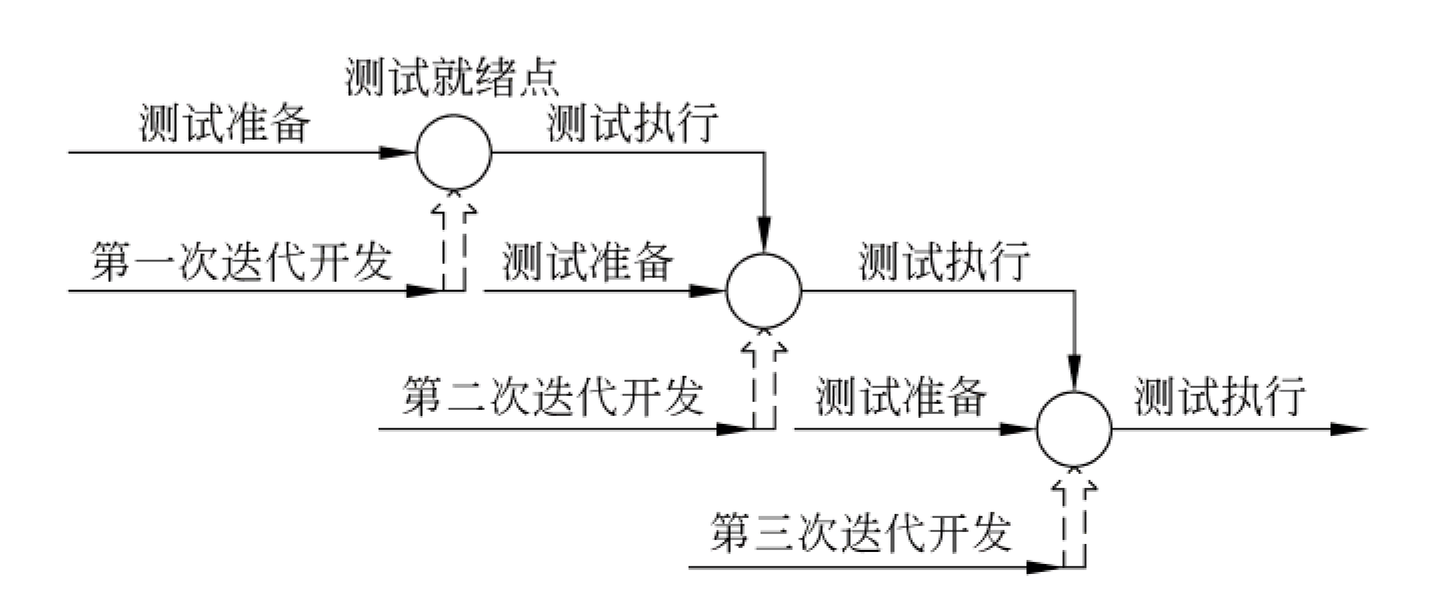


图 2-4　软件开发与系统测试的关系

H 模型使我们对软件测试有了更进一步的认识：软件测试不仅指测试的执行，还包括很多其他活动，如测试的准备；软件测试是一个独立的流程，可贯穿到软件产品整个生命周期中的任一流程，与之并发地进行；只要某个测试达到准备就绪点，测试执行活动就可以开展；不同的测试活动可以是按照某个次序先后进行的，但也可能是反复的。

4. 其他模型

除上述几种常见模型外，业界还流传着其他模型，例如 X 模型、前置测试模型等。

X 模型提出针对单独的程序片段进行相互分离的编码和测试，再通过频繁的交接，通

过集成最终合成为可执行的程序。

前置测试模型体现了开发与测试的紧密结合,要求对每一个交付内容进行测试。使用该模型可以加快项目进度。

2.3.2 测试过程的实施策略

1. 测试过程模型选取策略

以上介绍的测试过程模型为人们提供了软件测试的流程和方法,为测试过程管理提供了重要依据,对指导开展测试工作具有极重要的意义。但任何测试模型都不是万能的,不可能有哪种模型完全适用于某项测试工作。应针对具体的开发项目灵活地选择测试过程模型,尽可能地利用各模型中对项目有实用价值的方面,不能为使用模型而使用模型。

一般说来,在实际的测试活动中,可以 W 模型为框架,及早全面地开展测试,同时灵活地运用 H 模型的独立测试思想,将测试与开发过程紧密结合,在达到恰当的测试就绪点时执行独立的测试工作,测试工作应是可迭代的。

2. 测试过程管理理念

测试过程管理理念是从无数的测试实践中提炼出来的,能指导测试人员成功地策划和开展测试过程,对于测试人员是不可或缺的精神财富。测试过程管理牵涉的范围非常广泛,包括过程定义、人力资源管理、风险管理等,以下仅介绍从测试过程模型中提炼出来的、对实际测试有指导意义的测试过程管理理念。

(1) 尽早测试

“尽早测试”是从 W 模型中抽象出来的理念。测试不应是在编码之后才开展的工作,测试与开发是两个相互依存的、并行的过程,在开发过程中的早期——需求分析阶段就应开展测试工作。

“尽早测试”主要包含两方面的含义,第一,测试人员早期参与到软件项目中,及时开展测试的准备工作,包括编写测试计划、制定测试方案以及准备测试用例。第二,尽早开展测试执行工作,一旦代码模块完成,就应该及时开展单元测试;一旦代码模块被集成为相对独立的子系统,便可以开展集成测试;一旦有产品提交,便可以开展系统测试工作。

尽早开展测试准备工作,使测试人员能够在早期了解测试的难度,预测测试的风险,有利于制定出完善的测试计划和方案,提高软件测试及开发的效率,规避测试中存在的风险。尽早开展测试执行工作,有利于测试人员尽早地发现软件中的缺陷,大大降低了错误修复的成本。

需要注意的是,“尽早测试”并不是盲目地提前测试活动,测试活动开展的前提是达到相应的测试就绪点。

(2) 全面测试

软件是程序、数据和文档的集合,因而软件测试不仅仅是对程序的测试,还应包括对软件副产品的全面测试,这是 W 模型中一个重要的理念。需求分析文档、设计文档作为软件的阶段性产品,直接影响到软件的质量。大量实践表明,软件中的大部分错误不是在编码阶段而是在编码之前的需求分析和设计中造成的。

“全面测试”主要包含下面两方面的含义:

① 对软件的所有阶段性产品进行全面测试，包括需求分析文档、设计文档、程序代码、用户文档等。

② 软件开发人员和测试人员（有时还包括用户）均应参与到测试工作中，例如对需求的验证和确认活动，就需要开发人员、测试人员及用户的共同参与，这样才能保证软件最大限度地满足用户的需求。

“全面测试”有助于从多方面保证软件产品的质量。

（3）全过程测试

W 模型中体现的另一个测试理念就是“全过程测试”。W 模型的双 V 字形象地表明了软件开发与软件测试的紧密结合。

“全过程测试”主要包含下面两方面的含义：

① 测试人员要充分关注开发过程，对开发过程的各种变更及时做出响应。例如，根据开发进度计划的变更及时调整测试进度和测试策略，依据需求的变更及时补充和完善测试用例。

② 测试人员要对测试的全过程进行跟踪。例如，建立完善的度量和分析机制（软件度量将在第 10 章详细介绍），通过对自身过程的度量，及时了解测试过程信息，调整测试策略，持续改进测试过程。

全过程测试有助于及时应对项目开发过程中发生的变更，规避测试风险，且利于更好地把握和改进测试过程，优化测试活动。

（4）独立的、迭代的测试

独立的、迭代的测试是 H 模型倡导的理念。软件开发瀑布模型中严格的阶段划分只是一种理想状况，大多数的项目开发都存在需求分析、设计、编码等活动的迭代，故与它们相对应的测试工作也应是可迭代的。

W 模型强调软件测试与软件开发应紧密结合；H 模型在此基础上进一步揭示，测试与开发活动的紧密结合并不意味着测试是依附于开发活动的一个过程，测试活动应是独立的，可从开发活动中剥离出来的。测试应成为一个独立的流程，只要测试准备工作完成，达到了相应的测试就绪点，测试的执行活动就可以开展。

独立的、迭代的测试有利于应对开发过程复杂多变的情况，使测试工作更加灵活，与开发活动也结合得更紧密，因而非常有利于全过程测试。

2.4　测试驱动开发

2.4.1　测试驱动开发的概念

测试驱动开发（Test-Driven Development，TDD）是极限编程（eXtreme Programming，XP）的一个重要组成部分。XP 包含 12 个团队实践，TDD 是其中之一。

TDD 的基本思路是通过测试推动整个开发的进行。也就是在明确要实现某个功能的开发后，首先思考如何对这个功能进行测试，并完成测试代码的编写；接着编写相关的代码满足这些测试用例；然后循环添加其他功能，直到完成全部功能的开发。

测试驱动开发的基本步骤如下：

① 明确当前要完成的功能点(所有功能点记录在测试列表中)。

② 快速完成针对此功能的测试用例编写。

③ 编译测试代码,结果为通不过。

④ 编写对应的功能代码。

⑤ 测试代码编译通过。

⑥ 对代码进行重构,并保证全部测试代码运行通过。

⑦ 循环完成所有功能的开发。

使用 TDD 进行开发的目的之一是使开发更好地适应需求。需求向来就是软件开发过程中最不好明确描述且易变的东西,这里所说的需求不仅是指用户的需求,还包括对代码的使用需求。最令开发人员感到困难的就是到了开发后期还需要对某个类或者函数的接口进行修改或者扩展,发生这类事情的原因就在于对这部分功能代码的使用需求没有清晰的、可验证的描述。

测试驱动开发并不事先编写某功能对应的代码,而是先编写该功能代码的测试用例,对该功能的分解、使用过程、接口进行设计,以便很好地描述对功能代码的使用需求。这种描述清晰、可验证,提高了功能代码的内聚性和复用性,通常更符合后期开发的需求和可测试的要求。因此,测试驱动开发也是一个代码设计的过程。使用 TDD 后,设计、编码和测试三者之间的鸿沟将不再存在,促进了开发(测试)人员更好地理解整个项目环境。

TDD 可概括为三方面的活动:编写测试用例,编写功能代码并进行测试,以及重构代码来提高代码质量(使代码更简洁、灵活且易于理解)。这个过程将会循环出现,每一次循环,都会对设计和编码进行优化,提升软件的质量。

大多数设计都是自上而下创建的,并基于可观察的特性进行分类,促进对问题的理解和解决。换句话说,人们试图创建层次结构的对象,它模型化了问题域。但 TDD 与之不同,它有利于自下而上演进为设计,即通过对一些小问题依次实施一系列简单的解决方案,最终形成设计方案。为保证这种设计方案未偏离正确的路线,TDD 中每次循环的最后阶段要求进行重构,重构能发现设计方案中的错误,并对设计进行优化。

值得注意的是,在软件开发过程中,TDD 的应用领域并不仅限于代码编写阶段,还可以应用于整个开发过程的其他各阶段,如需求分析阶段、设计阶段等。对开发过程中的某阶段进行测试驱动时,首先应该思考如何对该阶段进行测试、验证、考核,并编写相关的测试文档或测试代码,然后开始下一步工作,最后再验证相关的工作。在后面的文字中,TDD 主要指用于编码阶段的。

2.4.2 测试驱动开发的优点

考虑实施 TDD 的最初目的主要是,更好地适应需求及需求变更,简化开发过程和缩短开发周期。

当然,TDD 带来的好处远不止这些,其优点还包括如下几点:

- 项目进度可预测。而传统的方式很难知道什么时候编码工作结束。
- 大部分时间代码处在高质量状态,100%的时间里成果是可见的。
- 提供了全面正确地认识代码和利用代码的机会,传统的开发方式则没有这个机会。

- 为利用已有成果的人提供 Sample，无论是要利用源代码，还是重用组件。
- 系统可以与详尽的测试集一起发布，从而对软件将来版本的扩展提供方便。
- 将设计、编码、测试融为一体。由于编写测试和编写功能代码的是相同的程序员，降低了理解代码所花费的成本；同时避免了设计角色，因为对于一个使用 TDD 技术的开发小组而言，每个人都在进行设计。
- 降低了开发小组间的交流成本，提高了相互信赖程度。
- 由于能清晰、无二义地描述对代码的需求，从而减少了文档对代码需求描述不清而引入 Bug 的可能。
- 在预先设计和紧急设计之间建立一种平衡点，为开发（或测试）人员区分哪些设计应该事先做，哪些设计应该迭代时做提供了一个可靠的判断依据，即避免了过度设计。
- 有利于发现比传统测试方式更多的 Bug。
- 使 IDE 的调试功能失去意义，避免了令人头痛的调试。

2.4.3 测试驱动开发的原则

对测试驱动开发的概念有较清楚的认识后，在实施 TDD 的过程中还应注意遵循如下原则：

① 测试隔离。不同代码的测试应该相互隔离。对一块代码的测试只考虑此代码的测试，不要考虑其实现细节，例如它是否使用了其他类的边界条件。

② 专注于当前工作。开发人员在开发过程中要做不同的工作，如编写测试代码、开发功能代码、对代码重构等。开发人员完成对应的工作时应该保持注意力集中在当前工作上，而不要过多地考虑其他方面的细节，无谓地增加工作复杂度。

③ 防止过度设计。编写功能代码时应该关注于完成当前功能点，通过测试，使用最简单、直接的方式编码。过多地考虑后期的扩展、其他功能的添加，则会增加问题的复杂性，反而容易产生问题。实际上，在要添加这些特性时，有整套测试用例做基础，通过不断重构是较容易实现的。

④ 及时补充测试列表。在开发过程中，对软件功能点的需求可能不断变更和增加。在任何阶段若需添加某功能需求时，应当把该功能点加到测试列表中，再继续手头工作。这样可以避免功能的疏漏，也避免干扰当前进行的工作，待到以后的循环中再完成刚添加进测试列表的功能点。

⑤ 先写断言（Assertion）。测试代码编写时，应该首先编写用于对功能代码的判断的断言语句，然后再编写相应的辅助语句。

⑥ 及时重构。代码简洁可用是测试驱动开发所追求的主要目标，其实这是为了优化设计和编码。无论是功能代码还是测试代码，对代码中存在的重复、结构不合理等情况，在测试通过后，应及时进行重构加以消除。

应该把重构看成一种书写代码的方式或习惯，重构随时都有可能发生。在测试驱动开发中，除去编写测试用例和实现测试用例之外的所有工作都是重构。所以，没有重构任何设计都不能实现。至于什么时候重构，可遵照如下的思路：

① 实现测试用例后重构代码；

② 完成某个特性后重构设计；

③ 产品的重构完成后对测试用例重构。

关于测试驱动开发实例，可参见 5.5.3 小节。

2.4.4 关于测试驱动开发的一些讨论

(1) 是写测试代码还是写测试文档

测试代码非常简单，通常是围绕某个情况的正确性判断的几个语句。如果功能点太复杂，就应该对其进行分解，直至可用简单的测试代码来表达需求。

传统的开发过程通常强调的是用测试文档来指导测试。但随着软件规模的日渐庞大，开发节奏的不断加快，用户需求的复杂多变，维护高层(需求分析、概要设计)的测试文档是可行的，但低层的测试文档(如详细设计及编码阶段的测试文档)的维护成本则太高。

因而，在详细设计阶段及编码阶段，应通过测试代码取代测试文档来驱动开发。这样做并不完全是出于节约成本的考虑。实际上，清晰、可实时验证功能正确性的测试代码就是对功能代码最好的测试文档。

(2) 何时停止编写测试用例(即测试的粒度问题)

测试驱动开发的思想强调测试不应该是开发过程中的负担，而是帮助我们减轻工作量的方法。何时停止编写测试用例，应根据测试人员的经验而定，复杂、核心的功能代码应该编写更全面、细致的测试用例。

对大的功能点测试和实现时，可先拆分成更小的功能点进行测试。比如，一个类 A 使用了类 B,C，就应先完成对 B,C 的测试和开发。但这并不意味着每个小类或者小函数都应该测试。测试人员应该运用自己的经验，对那些可能出问题的地方重点测试，其他地方等它真正出问题时再补测试代码即可。

(3) 何时设计

许多人在依照 TDD 开发软件时常被这个问题困扰，总是觉得有些问题应该在写测试用例之前定下来，而有些问题应该在新增测试用例的过程中自然出现。

实际上，设计和测试在 TDD 中没有严格的先后和界限。如前所述，TDD 机制有利于自下而上演进为设计，即通过对一些小问题，依次实施一系列简单的解决方案，最后形成设计方案；再通过重构保证设计是正确的、被优化的。

当然，这并不是说，设计在 TDD 中是完全不需要的，也不意味着在 TDD 中设计完全是由测试衍生出的。

TDD 中设计的时机应该由开发者自己把握，不要受到 TDD 方式的限制。但是，不需要事先设计的方面一定不要马上去考虑，以免增加问题的复杂性。也就是说，应避免过度设计。

(4) 为一个特性编写测试用例还是为一个类编写测试用例

虽然 TDD 的说明书上说应该为一个特性(feature)编写相应的测试用例，但为什么著名的 TDD 大师所写的测试用例都是与类/方法一一对应的呢？

为避免设计上的重大失误，通常人们针对特性编写测试用例，但若某个特性无法用测试用例表达，则会将这个特性细分，直到可以为细分后的特性写出测试用例为止。

之后，随着不断地重构代码及测试用例，不断地依据 TDD 进行开发，最后当产品伴

随测试用例集一起发布的时候，经过重构以后的测试用例很可能就是与产品中的类/方法一一对应的。

(5) 何时应将全部测试都运行一遍

测试驱动开发的思想要求我们在每次重构之后都应完整地运行全部测试用例，因为重构很可能会改变整个代码的结构或设计，使原来通过的测试用例现在不能通过，也就是说可能导致不可预见的后果。但有两种情况会使每次重构之后都完整运行全部测试用例的要求难以实现。这是因为正在开发的是一个大型的项目（如 ERP 系统），所有测试用例运行一遍可能将花费数个小时；并且重构由工具来完成（现在的情况大都如此），不由人所控制。

基于这两种情况，在条件不允许时，可以挑选几个我们认为可能受到本次重构影响的测试用例去运行。

(6) 什么场合不适合使用 TDD

对软件质量要求极高的军事或科研产品、人命关天的软件等不适用 TDD。

2.5　软件测试的原则

为了尽可能发现软件中的错误，提高软件产品的质量，在软件测试的实践中，应把握的测试原则主要有如下几条：

- 应尽早、全面、全过程、独立地开展测试活动。
- 所有的测试标准应建立在满足用户需求的基础上，软件中的最严重错误是那些导致用户需求无法满足的错误。
- 程序员应避免检查自己的程序，避免因心理因素造成的一系列不良影响，毕竟否认自己的工作成果是件不愉快的事。
- 设计测试用例时，应该考虑合法的输入和不合法的输入，以及各种边界条件；特殊情况下要制造极端状态和意外状态，以检验软件在各种可能的情况下能否正常工作。
- 要充分注意测试中错误的集中发生现象，这与程序员的编程习惯有很大的关系。
- 对测试中发现的错误应当有一个确认的过程，严重的错误可以召开评审会进行讨论和分析。
- 制定严格的测试计划并执行之，杜绝测试的随意性，并把测试时间安排得尽量宽松，不要希望在极短的时间内完成一个高水平测试。
- 充分重视回归测试，防止因出现修改一个错误或新增一个功能而造成软件出错的情况。
- 妥善保存一切测试文档。测试文档能对测试工作进行指导和提供评价，并为后续的测试工作提供依据。

2.6 软件测试文档

2.6.1 软件测试文档概述

软件测试贯穿于开发的全过程,是一项复杂的工作。应当把测试也当作一个项目,对其进行有效的计划和管理。软件测试文档(Software Test Documentation)为测试项目的组织、规划和管理提供了一个架构。

在测试的前期和测试过程中都需建立相应的测试文档,并应根据需求的变更及时调整测试文档。

软件测试文档主要包括测试计划、测试设计、测试用例、测试规程、测试事件报告、测试总结报告等。

为了统一测试文档的书写标准,IEEE/ANSI 制定了 829—1983 标准(修订版为 829—1998 标准)。还有其他的一些标准也用于指导软件测试文档的编写,如我国制定的《计算机软件测试文件编制规范》(GB/T 9386—1988)。

标准化的测试文档如同一种通用的参照体系,可以达到便于交流的目的。文档中所规定的内容可以作为对测试过程完备性的对照检查表,故有助于提高测试过程中每个阶段的能见度,极大地提高测试工作的可管理性。

2.6.2 《计算机软件测试文件编制规范》(GB/T 9386—1988)简介

下面对我国制定的《计算机软件测试文件编制规范》(GB/T 9386—1988)(以下简称规范)进行简要介绍。

1. 引用标准

该规范的引用标准为:

- GB/T 11457 软件工程术语;
- GB 8566 计算机软件开发规范;
- GB 8567 计算机软件产品开发文件编制指南。

2. 关键术语定义

对该规范中使用的关键术语定义如下。

- 设计层 (Design Level)

软件项的设计分解,如系统、子系统、程序或模块。

- 通过准则 (Pass Criteria)

一个软件项或软件特性的测试是否通过的判别依据。

- 软件特性 (Software Feature)

软件项的显著特性(如功能、性能或可移植性等)。

- 软件项 (Software Item)

源代码、目标代码、作业控制代码、控制数据或这些项的集合。

- 测试项 (Test Item)

作为测试对象的软件项。

3. 规范的主要内容

该规范确定了各个测试文件的格式和内容，所提出的文件类型包括测试计划、测试说明和测试报告。

测试计划（Test Plan）描述测试活动的范围、方法、资源和进度。它规定被测试的项、被测试的特性、应完成的测试任务、担任各项工作的人员职责及与本计划有关的风险等。

测试说明包括以下 3 类文件：

(1) 测试设计（Test Design）说明。详细描述测试方法，规定该设计及其有关测试所包括的特性，还规定完成测试所需测试用例和测试规程，并规定特性的通过准则。

(2) 测试用例说明。列出用于输入的具体值及预期的输出结果，并规定在使用具体测试用例时，对测试规程的各种限制。将测试用例与测试设计分开，可以使它们用于多个设计并能在其他情形下重复使用。

(3) 测试规程（Test Procedure）说明。规定对于运行系统和执行指定的测试用例来实现有关测试设计所要求的所有步骤。

测试报告（Test Report）包括以下 4 类文件：

(1) 测试项传递报告。指明在开发组和测试组独立工作的情况下或者在希望正式开始测试的情况下为进行测试而被传递的测试项。

(2) 测试日志。测试组用于记录测试执行过程中发生的情况。

(3) 测试事件报告。描述在测试执行期间发生并需进一步调查的一切事件。

(4) 测试总结报告。总结与测试设计说明有关的测试活动。

4. 对规范的实施

使用该规范的每个单位，要规定测试阶段所应有的特定文件，并在测试计划中规定测试完成后所能提交的全部文件。对于不同的设计层或不同规模的软件，所选文件的种类也可有所不同。

在所提供的每个标准文件中，每一章的内容对于具体的应用和特定的测试阶段可以有所增减。不仅可以调整内容，还可以在基本文件集中增加另外的文件。任何一个文件都可以增加新的内容，并且某章若无可写的内容，则可不写，但须保留该章编号。使用该规范的每个单位应该补充规定对内容的要求和约定，以便反映自己在测试、文件控制、配置管理和质量保证方面所用的特定方法、设备和工具。

图 2-5 反映了规范中各类测试文档之间的关系，同时也在一定程度上反映了测试的流程。

以下是规范中的文件编制实施和使用指南。

(1) 实施指南

在实施测试文件编制的初始阶段，可先编写测试计划与测试报告文件。测试计划为整个测试过程提供基础。测试报告将鼓励测试单位以良好的方式记录整个测试过程的情况。

经过一段时间的实践后，积累了一定的经验之后再逐步引进其他文件。测试文件编

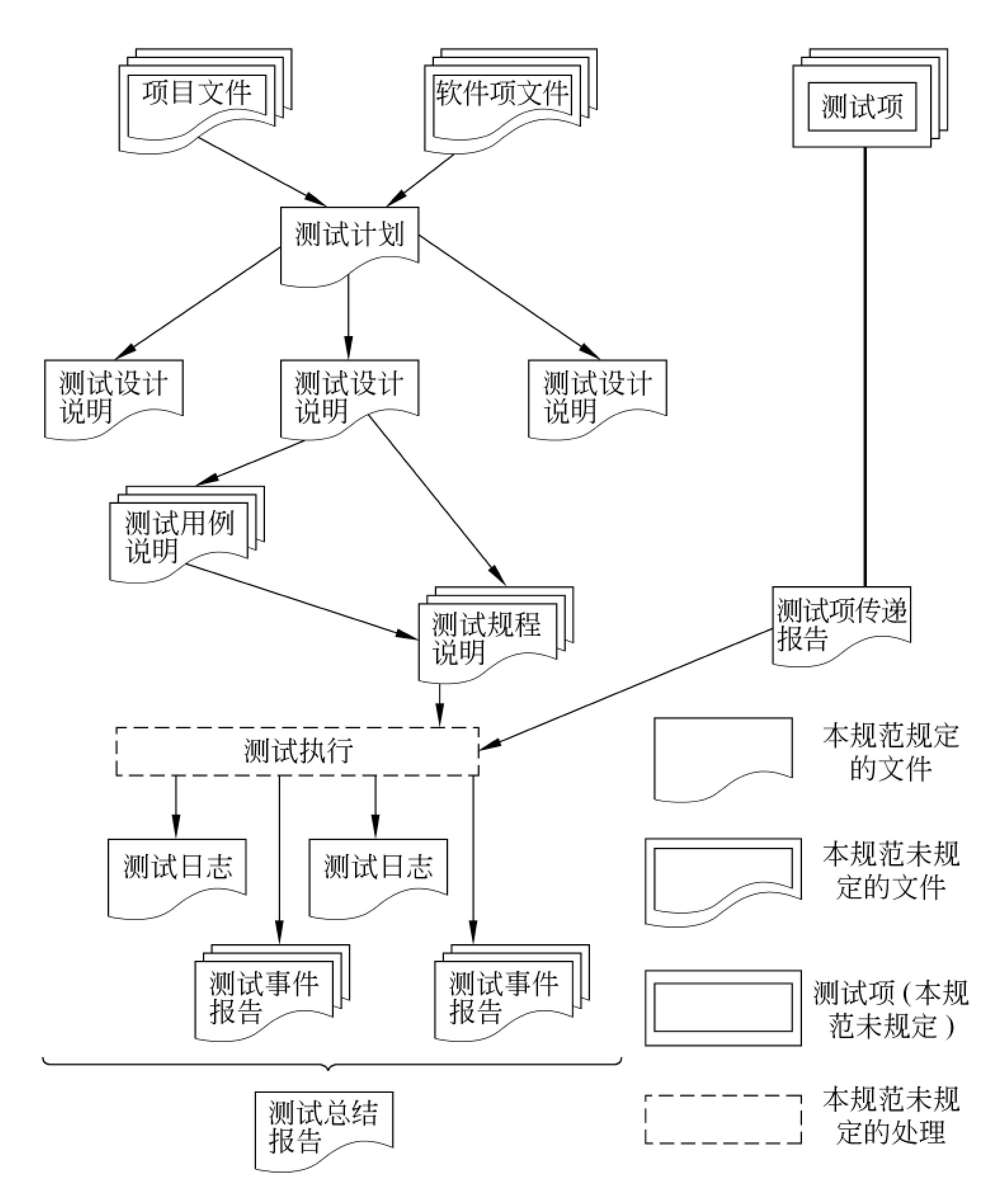


图 2-5　规范中各类测试文档之间的关系

制最终将形成一个相应于设计层的文件层次，即系统测试文件、子系统测试文件及模块测试文件等。在本单位所使用的、特定的测试技术的文件编制可以作为正文所述的基本文件集的补充。

（2）用法指南

在项目计划和单位标准中，应该指明在哪些测试活动中需要哪些测试文件，并可在文件中加入一些内容，使各个文件适应一个特定的测试项及一个特定的测试环境。

表 2-1 是一个在多种测试活动中所需测试文件的例子，所需文件数量将因单位而异。

表 2-1　一个测试文件的编制实例

	测试计划	测试设计说明	测试用例说明	测试规程说明	测试项传递报告	测试日志	测试事件报告	测试总结报告
验收	√	√	√	√	√	—	√	√
安装	√	√	—	—	√	—	√	√

续表

	测试计划	测试设计说明	测试用例说明	测试规程说明	测试项传递报告	测试日志	测试事件报告	测试总结报告
系统	√	√	√	√	√	√	√	√
子系统	—	—	√	√	√	√	√	√
模块	—	√	√	—	—	—	—	√

2.6.3　规范 GB/T 9386—1988 内容要求

以下是该规范中各个测试文件的书写格式及内容。对于每一个文件而言，都包含若干章，其内部各章应按指定的次序排列，补充的章可以放在最后或放在“批准”一章的前面（如果该文件最后一章是“批准”）。

如果某章的部分或全部内容在另一个文件中，则应在相应的内容位置上列出所引用的材料。引用的材料必须附在该文件后面，或交给文件的使用者。

1. 测试计划

测试计划共 16 章，结构如图 2-6 所示。

(1) 测试计划名称
(2) 引言
(3) 测试项
(4) 被测试特性
(5) 不被测试特性
(6) 方法
(7) 项通过准则
(8) 暂停标准和再启动要求
(9) 应提供的测试文件
(10) 测试任务
(11) 环境要求
(12) 职责
(13) 人员和训练要求
(14) 进度
(15) 风险和应急
(16) 批准

图 2-6　测试计划

每一章的详细内容如下。

(1) 测试计划名称(该计划的第 1 章)

为该测试计划取一个专用的名称。

(2) 引言(第 2 章)

归纳所要求测试的软件项和软件特性，可以包括系统的目标、背景、范围和引用材料等。

在最高层测试计划中，如果存在项目计划、质量保证计划、有关的政策、有关的标准等文件，则需要引用它们。

(3) 测试项(第 3 章)

描述被测试的对象，包括其版本、修订级别，并指出在测试开始之前对逻辑或物理变换的要求。

(4) 被测试特性(第 4 章)

指明所有被测试的软件特性及其组合，指明每个特性或特性组合有关的测试设计说明。

(5) 不被测试特性(第 5 章)

指出不被测试的所有特性和特性的有意义的组合及其理由。

(6) 方法(第 6 章)

描述测试的总体方法，规定测试指定特性所需的主要活动、技术和工具，详尽地描述方法，以便列出主要的测试任务，并估计执行各项任务所需的时间；规定所希望的最低程度的测试彻底性，指明用于判断测试彻底性的技术，例如检查哪些语句至少执行过一次；指出对测试的主要限制，例如测试项可用性、测试资源的可用性和测试截止期限等。

（7）项通过准则（第 7 章）

规定各测试项的测试标准。

（8）暂停标准和再启动要求（第 8 章）

规定用于暂停全部或部分与该计划有关的测试项的测试活动标准；规定测试再启动时必须重复的测试活动。

（9）应提供的测试文件（第 9 章）

规定测试完成后所应递交的文件，这些文件可以是前述 8 个文件的全部或者部分。

（10）测试任务（第 10 章）

指明执行测试所需的任务集合，指出任务间的一切依赖关系和所需的一切特殊技能。

（11）环境要求（第 11 章）

规定测试环境所必备的和希望的性质，包括硬件、通信和系统软件的物理特征、使用方式以及任何其他支撑测试所需软件或设备；指出所需的特殊测试工具及其他测试要求，例如出版物或办公场地等；指出测试组目前还不能得到的所有要求的来源。

（12）职责（第 12 章）

指出负责管理、设计、准备、执行、监督、检查和仲裁的小组，以及负责提供测试计划中的测试项和测试计划中的环境要求的小组。

这些小组可以包括开发人员、测试人员、操作员、用户代表、数据管理员和质量保证人员。

（13）人员和训练要求（第 13 章）

指明测试人员应有的水平，以及为掌握必要技能可供选择的训练科目。

（14）进度（第 14 章）

包括在软件项目进度中规定的测试里程碑，以及所有测试项传递时间。

定义所需的新的测试里程碑，估计完成每项测试任务所需的时间，为每项测试任务和测试里程碑规定进度，对每项测试资源规定使用期限。

（15）风险和应急（第 15 章）

预测测试计划中的风险，规定对各种风险的应急措施，例如延期传递的测试项可能需要加夜班来赶上规定的进度。

（16）批准（第 16 章）

规定该计划的审批者（姓名和职务），为签名和填写日期留出空间。

2. 测试设计说明

测试设计说明共 5 章，如图 2-7 所示。

（1）测试设计说明名称
（2）被测试特性
（3）方法详述
（4）测试用例名称
（5）特性通过准则

图 2-7 测试设计说明

下面给出每一章的详细内容。

（1）测试设计说明名称（该说明的第 1 章）

给每一个测试设计说明取一个专用名称。如果存在，也可以引用有关的测试计划中给出的名称。

（2）被测试特性（第 2 章）

规定测试项，描述作为该设计测试目标的特性和特性的组合，其他特性可以论及，但

不必测试。

（3）方法详述（第 3 章）

将测试计划中规定的方法进行细化，包括要用的具体测试技术，规定分析测试结果的方法（如比较程序或人工观察）。

规定为选择测试用例提供合理依据的一切分析结果。例如，可以说明容错的条例（如区别有效输入和无效输入的条件）。

归纳所有测试用例的共同属性，包括输入约束条件、对共享环境的要求、对共享的特殊规程的要求及任何共享的测试用例间的依赖关系。

（4）测试用例名称（第 4 章）

列出与该测试设计说明有关的每一个测试用例的名称和简要说明，其中某个特定的测试用例可能在多个测试设计说明中出现。

（5）特性通过准则（第 5 章）

规定用于判别特性和特性组合是否通过测试的准则。

3. 测试用例说明

测试用例说明共 7 章，结构如图 2-8 所示。由于测试用例可能被由多个小组长期使用的多个测试设计说明引用，所以在测试用例说明中必须包含足够具体的信息以便重复使用。

（1）测试用例说明名称
（2）测试项
（3）输入说明
（4）输出说明
（5）环境要求
（6）特殊的规程要求
（7）用例间的依赖关系

图 2-8　测试用例说明

下面给出每一章的详细内容。

（1）测试用例说明名称（第 1 章）

给该测试用例说明取一个专用名称。

（2）测试项（第 2 章）

规定并简要说明该测试用例所涉及的项和特性，对于每一项，可以考虑引用以下文件：需求说明书、设计说明书、用户手册和操作手册。

（3）输入说明（第 3 章）

规定执行测试用例所需的输入。有些输入可以用值（允许适当的误差）规定；而另一些输入，如常数表或事务文件可以用名来规定。规定所有合适的数据库、文件、终端信息、内存常驻区域和由操作系统传送的值，规定各输入间所需所有关系（如时序关系等）。

（4）输出说明（第 4 章）

规定测试项的所有输出和特性，例如响应时间。提供各个输出或特性的正确值（在适当的误差范围内）。

（5）环境要求（第 5 章）

① 硬件。规定执行该测试用例所需硬件特征和配置。

② 软件。规定执行该测试用例所需系统软件和应用软件。系统软件可以包括操作系统、编译程序、模拟程序和测试工具等。

③ 其他。说明所有其他要求，如特种设施要求或经过专门训练的人员等。

(6) 特殊的规程要求(第 6 章)

描述对执行该测试用例的测试规程的一切特殊限制。这些限制可以包括特定的准备,操作人员的干预,确定特殊的输出和清除过程。

(7) 用例间的依赖关系(第 7 章)

列出必须在该测试用例之前执行的测试用例名称,归纳依赖性质。

4. 测试规程说明

测试规程说明共 4 章,结构如图 2-9 所示。

下面给出每一章的详细内容。

(1) 测试规程说明名称
(2) 目的
(3) 特殊要求
(4) 规程步骤

图 2-9 测试规程说明

(1) 测试规程说明名称(该说明的第 1 章)

给每个测试规程说明取一个专用名称,给出对有关测试设计说明的引用。

(2) 目的(第 2 章)

描述该规程的目的。如果该规程执行测试用例,则引用各有关的测试用例说明。

(3) 特殊要求(第 3 章)

指出执行该规程所需的所有特殊要求,包括作为先决条件的规程、专门技能要求和特殊环境要求。

(4) 规程步骤(第 4 章)

① 日志。说明用来记录测试的执行结果,观察到的事件和其他与测试有关的事件(见测试日志和测试事件报告)的所有特殊方法或格式。

② 准备。描述新任务执行规程所必需的动作序列。

③ 启动。描述开始执行规程所必需的动作。

④ 处理。描述在规程执行过程中所必需的动作。

⑤ 度量。描述如何进行测试度量。

⑥ 暂停。描述因发生意外事件暂停测试所必需的动作。

⑦ 再启动。规定所有再启动点和在启动点上重新启动规程所必需的动作。

⑧ 停止。描述正常停止执行时所必需的动作。

⑨ 清除。描述恢复环境所必需的动作。

⑩ 应急。描述处理执行过程中可能发生的异常事件所必需的动作。

5. 测试项传递报告

测试项传递报告共 5 章,结构如图 2-10 所示。

下面给出该报告每一章的详细内容。

(1) 传递报告名称(该报告的第 1 章)

为该测试项传递报告取一个专用名称。

(2) 传递项(第 2 章)

规定被传递的项及其版本/修订级别;提供与传递项有关的项文件和测试计划的相关信息,指出对该传递项负责的人员。

(3) 位置(第 3 章)

规定传递项的位置及其所在媒体。

(4) 状态(第 4 章)

描述被传递的测试项的状态,包括其与项文件、这些项的以往传递及测试计划的差别;列出希望由被传递项解决的事件报告。

(5) 批准(第 5 章)

规定该传递报告必须审批者(姓名和职务),并为签名和日期留出空间。

6. 测试日志

测试日志共 3 章,如图 2-11 所示。

(1) 传递报告名称
(2) 传递项
(3) 位置
(4) 状态
(5) 批准

图 2-10　测试项传递报告

(1) 测试日志名称
(2) 描述
(3) 活动和事件条目

图 2-11　测试日志

下面给出每一章的详细内容。

(1) 测试日志名称(该日志的第 1 章)

为该测试日志取一专用的名称。

(2) 描述(第 2 章)

除了在日志条目中特别注明的以外,用于日志中所有条目的信息都包括在这一章中。应该考虑有以下信息:

- 规定被测试项及其版本/修订级别。如果存在,引用各项的传递报告。
- 规定完成测试的环境属性,包括设备说明、所用硬件、所用系统软件及可用存储容量等可用资源。

(3) 活动和事件条目(第 3 章)

对每个事件(包括事件的开始和结束),记录发生的日期和时间,并说明记录者。应考虑以下各项信息:

- 执行描述。记录所执行的测试规程的名称,并引用该测试规程说明。记录执行时在场人员包括测试者、操作员和观察员,还要说明每个人的作用。
- 测试结果。对每次执行,记录人工可观察到的结果(如产生的错误信息、异常中止和对操作员动作的请求等),以及所有输出的位置(如磁带号码),测试的执行是否成功。
- 环境信息。记录该条目的一切特殊的环境条件。
- 意外事件。记录意外事件及其发生前后的情况,例如请求显示总计,屏幕显示正常但响应时间似乎异常地长,重复执行时响应时间也同样过长。记录无法开始执行测试或无法结束测试的周围环境,例如电源故障或系统软件问题。
- 事件报告名称。每产生一个测试事件报告时记录其名称。

7. 测试事件报告

测试事件报告共4章，结构如图2-12所示。

下面给出每一章的详细内容。

(1) 测试事件报告名称(该报告的第1章)

为该测试事件报告取一个专用名称。

(2) 摘要(第2章)

简述事件，指出有关测试项及其版本/修订级别。引用有关的测试规程说明、测试用例说明和测试日志。

(1) 测试事件报告名称 (2) 摘要 (3) 事件描述 (4) 影响

图2-12 测试事件报告

(3) 事件描述(第3章)

对事件进行描述。该描述应包括：

输入、预期结果、实际结果、异常现象、日期和时间、规程步骤、环境、重复执行的意图、测试者和观察者。

该描述应该包括有助于确定事件发生的原因，以及改正其中错误的有关活动及观察。例如，描述可能对此事件有影响的所有测试用例的执行情况，描述与已公布的测试规程之间的一切差异等。

(4) 影响(第4章)

在所知道的范围内指出该事件对测试计划、测试设计说明、测试规程说明或测试用例说明所产生的影响。

8. 测试总结报告

规定该报告的审批者(姓名和职务)，并为签名和日期留出空间。

2.7 小结

本章阐述了软件测试的一系列基本概念、原则，介绍了软件测试文档的作用、分类及编写规范，是学习后续章节的基础。首先介绍了软件测试的定义、对象及著名的V&V，接着从不同角度介绍了软件测试的分类，讨论了几个重要的软件测试过程模型，并阐述了测试过程管理的若干理念，以及测试驱动开发的思想和原则，指出了软件测试的若干原则，最后介绍了软件测试文档的作用、分类和编写规范。

在测试分类中介绍的一些主要测试方法和技术，如白盒测试、黑盒测试、性能测试等，是本书的重要内容，将在后续章节详细介绍。

习 题

1. 软件测试的定义是什么？它与以前理解的定义有何不同？
2. 软件测试的对象仅仅是程序吗？谈谈你的观点。
3. V&V中两个V的含义分别是什么？它们属于测试吗？两者有何区别？
4. 按开发阶段，软件测试可分为哪几类？

5. α 测试和 β 测试的含义分别是什么？两者有何区别和联系？

6. 白盒测试、黑盒测试、灰盒测试三者有何区别？

7. 什么是软件测试过程？

8. 软件测试 V 模型有何不足之处？

9. 将测试活动和开发活动紧密结合是哪种软件测试模型所倡导的？这样做的好处是什么？

10. W 模型有何局限性？为什么？

11. H 模型倡导的理念是什么？它为何能弥补 W 模型的不足？

12. H 模型是否有利于“全过程测试”？为什么？

13. 测试驱动开发的含义是什么？它的基本步骤是怎样的？

14. 你认为测试驱动开发有哪些显著的优点？为什么？

15. 进行软件测试时应把握的原则主要有哪些？

16. 软件测试文档在软件测试中起什么样的作用？其主要包括哪几类？

第 3 章

黑盒测试方法

本章要点：

- 测试用例的含义。
- 设计测试用例的基本准则。
- 黑盒测试的含义。
- 黑盒测试无法实现穷举测试的原因。
- 黑盒测试的优点和局限性。
- 各种典型的黑盒测试方法。
- 黑盒测试方法的综合使用策略。

在这一章，将介绍测试用例的含义和设计准则，黑盒测试方法的概念和特点，以及典型的黑盒测试方法，主要包括等价类划分法、边界值分析法、错误推测法、因果图法、判定表法、正交试验法、场景法、功能图法等，通过对其实施策略和具体范例的介绍，使读者对这些方法有较深刻的理解，能够在今后的测试实践中成功运用。最后介绍了黑盒测试方法的综合使用策略。

3.1　测试用例

1. 测试用例的概念

测试用例(Test Case)是为特定目标开发的一组测试输入、执行条件和预期结果，其目的是测试程序中的某路径，核实程序或软件能否完成某个特定的功能需求。

程序在执行某测试用例的输入数据时，若其输出结果与测试用例中的预期结果不同，则说明程序中存在缺陷。

完成对一个被测程序的测试一般需要设计并执行多个测试用例。

2. 测试用例的重要性

确定测试用例是很重要的，原因有以下几个方面：

(1) 测试用例构成了设计和制定测试过程的基础。

(2) 测试的“深度”与测试用例的数量呈比例。由于每个测试用例反映不同的场景、

条件或经由产品的事件流，因而随着测试用例数量的增加，开发人员及测试人员对产品质量和测试流程也就越有信心。

（3）判断测试是否完全的一个主要评测方法是基于需求的覆盖，而这又是以确定、实施和执行的测试用例的数量为依据的。类似下面这样的说明："95% 的关键测试用例已得以执行和验证"，远比"已完成 95% 的测试"更有意义。

（4）测试工作量与测试用例的数量呈比例。根据全面且细化的测试用例，可以更准确地估计测试周期各连续阶段的时间安排。

（5）测试设计和开发的类型及所需的资源主要都受控于测试用例。

3. 测试用例的设计方法概述

由于穷举测试是不可能的，故测试用例的编写是一项富于技巧性和挑战性的工作，是软件测试中的重点和难点。测试人员应从数量极大的可用测试用例中精心挑选数量有限的、具有代表性或特殊性的测试用例，以高效地揭露程序或软件中的错误。

设计测试用例的基本准则如下：

（1）测试用例的代表性。设计测试用例时，应尽量覆盖各种合理的和不合理的，合法的和非法的，边界的和越界的，以及极限的输入数据、操作和环境设置，设计的测试用例应是最有可能发现程序或软件中错误的。

（2）测试用例的非重复性。测试用例不应是与原有测试用例重复的，应追求测试用例数目的精简。

（3）测试结果的可判定性。测试执行结果的正确性是可判定的，每一个测试用例都应有相应的预期结果。

（4）测试结果的可再现性。对同样的测试用例，被测程序的执行结果应当是相同的。

一般来说，测试用例设计属于重复次数少的智能活动，不太适合于自动化。但也有一些场合可以进行一定程度的自动化以提高设计效率，实现这种目的的工具被称为测试输入生成工具。测试输入生成工具可产生大量的测试用例，但它不能区分哪些用例是最重要的，这项任务只能由测试人员完成。

工具生成测试用例依赖于规格描述的形式化，如果不能做到形式化描述，是无法按照一定算法实现用例设计自动化的。另外由于用例的生成依赖于所采用的算法，所以工具生成用例比人工设计更彻底、精确。但人工设计测试用例在判断用例覆盖功能点是否有遗漏方面更具有优势。

测试用例的设计方法主要分为白盒法、黑盒法和灰盒法。

3.2　黑盒测试方法概述

3.2.1　黑盒测试的概念和对象

1. 黑盒测试的概念

黑盒测试又称为数据驱动测试或基于规格说明的测试。"黑盒"可理解为程序或软件装在一个漆黑的盒子里，所以盒子内的程序对测试人员来说是不可见的。

执行黑盒测试的人员在完全不考虑程序或软件的内部逻辑结构和处理过程的情况

下，根据软件的需求规格说明书设计测试用例，在程序或软件的界面上进行测试。故黑盒测试是从用户角度出发进行的测试。

用黑盒方法设计测试图例如图 3-1 所示，每个测试用例包括输入数据和预输出数据，在执行测试用例时，将实际输出数据与预输出数据进行比较，两者若不同，则说明程序很可能存在缺陷。

程序或软件

输入数据 预期输出数据

图 3-1　黑盒测试用例设计思路

黑盒测试的目的主要是为了发现以下错误：

(1) 是否有不正确或遗漏了的功能。

(2) 在接口上输入能否正确地接受。能否输出正确的结果。

(3) 是否有数据结构错误或外部信息(例如数据文件)访问错误。

(4) 性能上是否能够满足要求。

(5) 是否有初始化或终止性错误。

很多人将黑盒测试称为功能测试，实际上，进行功能测试是黑盒测试的主要任务，但并不是其全部，黑盒测试还包括性能测试等。

黑盒方法不可能实现穷举测试，主要原因如下：

(1) 在测试某功能时不可能对其所有输入值进行测试，更不可能对其所有输入取值组合进行测试。

(2) 无法对需求规格说明书中未规定的潜在需求进行测试。

2. 黑盒测试的对象

在一个软件的开发过程中，其需要测试的功能分为不同的层次，单个程序模块有其规定的功能，而多个程序模块集成后作为一个整体亦有其规定的功能需求，而且模块的集成一般是一个持续的过程，会经过若干临时版本，直至形成最终软件系统。所以黑盒测试的对象既可以是单个程序，也可以是模块集成过程中的多个临时版本和最终软件。为简单起见，在下文中多处将黑盒测试方法的对象说成是“程序”。

3.2.2　黑盒测试的优点和局限性

1. 黑盒测试的优点

正由于黑盒测试是在程序或软件界面上进行的测试，故其具有两个优点：

(1) 黑盒测试不考虑程序或软件的具体实现，若程序或软件的内部实现发生了变化，原先的测试用例依然可用；

(2) 黑盒测试用例的设计可以与软件的实现同时进行，因而加快了软件测试与开发的速度。

2. 黑盒测试的局限性

但黑盒测试也存在一些局限性。

(1) 黑盒测试是从程序的界面上进行的测试，故有时难以查找出错误的具体原因和位置，还需要通过执行白盒测试来进行更细致的错误定位。

(2) 黑盒测试的唯一依据是软件的需求规格说明书，它无法发现需求规格说明本身

存在的问题。例如,由于考虑不周或需求规格说明书中陈述的需求缺乏清晰的层次,导致需求规格说明书未能全面、准确地表达用户的需求,都会使黑盒测试的有效性大打折扣。

3.3　典型的黑盒测试方法

典型的黑盒测试方法包括等价类划分法、边界值分析法、错误推测法、因果图法、判定表法、正交试验法、场景法、功能图法等,下面对这些方法分别加以介绍。

3.3.1　等价类划分法

1. 等价类划分法的实施策略

等价类划分法是一种典型的黑盒测试用例设计方法。采用等价类划分法时,完全不考虑程序的内部结构,设计测试用例的唯一依据是软件需求规格说明书。

所谓等价类,是被测程序输入域的一个子集合,该输入集合中的数据对于揭示程序中的错误是等价的。故对某等价类中的代表值的测试就相当于对等价类中所有值的测试,这大大减少了测试用例的数目,提高了测试的效率。

使用等价类划分法时,应仔细分析需求规格说明书,将被测程序的输入域划分为若干等价类。等价类又分为有效等价类和无效等价类。

所谓有效等价类,是指该等价类中的输入数据是符合需求规格说明的;而无效等价类是指该等价类中的数据是违反需求规格说明的。若使用强类型语言编程,无效输入会导致运行时出错,因此不需考虑无效等价类。

有效等价类和无效等价类都是使用等价类划分法设计测试用例时所必须的,因为被测程序若是正确的,就应既能接受有效输入数据,也能接受无效输入数据的考验。

划分等价类的若干原则如下:

(1) 在规定了输入数据的取值范围的情况下,可以确定一个有效等价类和两个无效等价类。如月份取值应在 1~12 之间,可由此确定一个有效等价类即月份取值在 1~12 之间,和两个无效等价类,即月份取值小于 1 及月份取值大于 12。

(2) 在规定了输入条件必须如何的情况下,可以确定一个有效等价类和一个无效等价类。如输入值必须大于 0,则有效等价类为输入值必须大于 0,无效等价类为输入值小于或等于 0。

(3) 在输入数据是一个布尔量的情况下,可以确定一个有效等价类和一个无效等价类。

(4) 在规定了输入数据由 n 个值构成,并要求对其中的每个值进行测试时,可以确定 n 个有效等价类和一个无效等价类。

(5) 在规定了输入数据必须遵守的规则的情况下,可以确定一个有效等价类和若干个无效等价类(从不同角度违反规则)。如规定输入值必须是数字类型的字符,则可确定一个有效等价类,即输入值为数字类型的字符,和多个无效等价类,即输入值为字母、为专用字符(如+、*、/等)及为非打印字符(如回车、空格等)。

(6) 在已划分的某等价类中,若各元素在程序中的处理方式不同,则应将此等价类进

一步划分为若干等价类。

划分等价类的过程是一个对软件的需求规格说明进行分析推敲的过程，有时还需要多次的尝试方能成功。

对被测程序划分出若干有效等价类和无效等价类之后，应建立等价类表，列出与每一输入条件对应的有效等价类和无效等价类，格式如表 3-1 所示。

表 3-1　等价类表格式

输入条件	有效等价类	无效等价类
…	…	…
…	…	…

接着根据等价类表编写测试用例，可遵循如下步骤：

① 为每一等价类规定一个唯一的编号。

② 设计一个测试用例，使其尽可能多地覆盖尚未被覆盖的有效等价类。如此重复，直至所有有效等价类均被测试用例所覆盖。

③ 设计一个测试用例，使其覆盖一个尚未被覆盖的无效等价类。如此重复，直至所有无效等价类均被测试用例所覆盖。

2. 等价类划分法举例

下面给出一个运用等价类划分法进行黑盒测试的例子。

试根据如下的规格说明，采用等价类划分方法给出足够多的测试用例。“一个程序读入 3 个整数，把这 3 个整数的值看作一个三角形的 3 条边的长度，根据 3 条边长确定该三角形是普通的、等腰的还是等边的，并打印出结论”。

可设三角形的 3 条边长分别为 A，B，C。若此 3 条边能构成一个普通三角形，必须满足

$$A>0,\quad B>0,\quad C>0,\quad A+B>C,\quad B+C>A,\quad A+C>B$$

若此 3 条边能构成一个等腰三角形，必须进一步满足

$$A=B,\quad \text{或 } A=C,\quad \text{或 } B=C$$

若此 3 条边能构成一个等边三角形，必须在满足为普通三角形的基础上进一步满足

$$A=B,\quad \text{且 } B=C,\quad \text{且 } A=C$$

通过对程序功能的分析，列出等价类表如表 3-2 所示。

表 3-2　判断三角形类型程序的等价类表

输入条件	有效等价类	无效等价类
是否为普通三角形	$A>0$ (1) $B>0$ (2) $C>0$ (3) $A+B>C$ (4) $B+C>A$ (5) $A+C>B$ (6)	$A=0$ (7) $B=0$ (8) $C=0$ (9) $A+B=C$ (10) $B+C=A$ (11) $A+C=B$ (12)

续表

输 入 条 件	有效等价类	无效等价类
是否为等腰三角形	$A=B$ (13) $B=C$ (14) $C=A$ (15)	$A\neq B$ and $B\neq C$ and $C\neq A$ (16)
是否为等边三角形	$A=B$ and $B=C$ and $C=A$ (17)	$A\neq B$ (18) $B\neq C$ (19) $C\neq A$ (20)

接下来设计测试用例，设输入数据为[A,B,C]，测试用例如表 3-3 所示。

表 3-3　判断三角形类型程序的基于等价类方法的测试用例

序号	输入数据	覆盖的等价类	预 期 输 出
1	[3,4,5]	(1),(2),(3),(4),(5),(6)	为普通三角形
2	[0,1,2]	(7)	不为三角形
3	[1,0,2]	(8)	
4	[1,2,0]	(9)	
5	[1,2,3]	(10)	
6	[3,1,2]	(11)	
7	[1,3,2]	(12)	
8	[3,3,4]	(1),(2),(3),(4),(5),(6),(13)	为等腰三角形
9	[3,4,4]	(1),(2),(3),(4),(5),(6),(14)	
10	[3,4,3]	(1),(2),(3),(4),(5),(6),(15)	
11	[3,4,5]	(1),(2),(3),(4),(5),(6),(16)	为非等腰三角形(为普通三角形)
12	[3,3,3]	(1),(2),(3),(4),(5),(6),(17)	为等边三角形
13	[3,4,4]	(1),(2),(3),(4),(5),(6),(14),(18)	为非等边三角形(为等腰三角形)
14	[3,4,3]	(1),(2),(3),(4),(5),(6),(15),(19)	
15	[3,3,4]	(1),(2),(3),(4),(5),(6),(13),(20)	

通过对表 3-3 的仔细分析，还可对表 3-3 中的测试用例进行如下简化：

- 第 11 号测试用例与第 1 号测试用例重复，可以略去，类似地，第 8 号和第 15 号测试用例、第 9 号和第 13 号测试用例、第 10 号和第 14 号测试用例也是分别相同的，故第 13,14,15 号测试用例也可略去。
- 若不考虑 A,B,C 的特定取值，第 2,3,4 号测试用例中选一个即可，类似地，第 5,6,7 号测试用例中选一个即可，第 8,9,10 号测试用例中选一个即可。

在对该判断三角形类型的程序执行表 3-3 中的测试用例过程中，若执行某测试用例

中的输入数据，得到的输出结果与测试用例中的预期输出(注意：完整的测试用例应包括预期输出)不同，如执行输入数据为[3,3,3]的测试用例，得到的输出为等腰三角形，则说明该程序中存在错误，需要进一步查找错误的原因和位置并进行修复。

3.3.2 边界值分析法

大量测试实践表明，很多错误是发生在输入或输出数据范围的边界上。因而针对各种边界情况设计测试用例，有利于揭露程序中的错误。

如 3.3.1 小节中的判断三角形类型的例题中，若此 3 条边能构成一个普通的三角形，必须满足 $A>0$，$B>0$，$C>0$，$A+B>C$，$B+C>A$，$A+C>B$；对于等价类 $A>0$，其边界是 $A=0$，可针对此边界设计测试用例，以验证程序在 $A=0$ 时的输出是否正确；若 $A=0$ 时程序的输出仍为普通三角形，则说明程序中存在错误。

边界值分析法是一种对等价类划分法的补充。使用边界值分析法时，应针对等于、刚好大于或刚好小于各输入等价类和输出等价类(此处对 3.3.1 小节中等价类的概念进行了扩展，但并不影响对实际问题的理解)边界值的情况设计测试用例。

用边界值分析法设计测试用例，应遵循如下原则：

(1) 若输入条件规定了值的范围，应针对刚达到此范围边界的值，以及刚超越此范围边界的值设计测试用例。

(2) 若输入条件规定了值的个数，针对最大个数、最小个数、比最大个数大 1、比最小个数小 1 等几种情况的数据设计测试用例。

(3) 若输出条件规定了值的范围，应针对刚达到此范围边界的值，以及刚超越此范围边界的值设计测试用例。

(4) 若输出条件规定了值的个数，针对最大个数、最小个数、比最大个数大 1、比最小个数小 1 等几种情况的数据设计测试用例。

(5) 若程序的规格说明中给出的输入域或输出域是有序集合，应针对集合的第一个元素(即刚好大于或小于第一个元素值的数值)和最后一个元素(即刚好大于或小于最后一个元素值的数值)设计测试用例。

(6) 若程序中使用了内部数据结构，应针对该内部数据结构边界上的值设计测试用例。

(7) 进一步分析规格说明，找出其他可能的边界情况，针对它们设计测试用例。

当然，运用以上原则时不能生搬硬套，最重要的是在分析具体规格说明的基础上寻找边界情况，进而设计测试用例。

关于边界值分析法的实例将在 3.3.3 小节中一并介绍。

3.3.3 错误推测法

1. 错误推测法的实施策略

所谓错误推测法(Error Guessing)，是根据测试人员的经验和直觉推测程序中可能存在的错误，有针对性地设计测试用例，可以验证事先的推测是否正确。

错误推测法一般是针对程序的敏感点设计测试用例。例如，人们总结了在单元测试时模块中容易发生的一些错误，可根据此经验设计测试用例；又如输入数据为空，或输入

表格为空，凭经验来看是程序不一定能正确处理的情况，可针对这些情况设计测试用例；若程序要求输入正数，可输入负数观察程序能否正确处理该情况；若程序对输入时间敏感，可将公元 4000 年作为输入数据，观察程序如何处理。

2. 边界值分析法和错误推测法举例

下面考查一个综合使用边界值分析法和错误推测法进行黑盒测试的范例。

有一个标准化考试阅卷程序，能批阅考生试卷并生成统计报告，其规格说明如下。

标准化考试阅卷程序的输入由若干记录组成，每个记录均含有 80 个字符；记录根据类型可分为 3 组，分别为标题记录、标准答案记录和学生答卷记录，如图 3-2 所示。

标题记录

标　题
1　　　　　　80

标准答案记录

试题数		标准答案(第 1～50 题)	2
1　3		10　59	80

试题数		标准答案(第 51～100 题)	2
1　3		10　59	80

⋮

学生答卷记录

学号 1	学生答案(第 1～50 题)	3
1　9	10　59	80

学号 1	学生答案(第 51～100 题)	3
1　9	10　59	80

⋮

学号 n	学生答案(第 1～50 题)	3
1　9	10　59	80

学号 n	学生答案(第 51～100 题)	3
1　9	10　59	80

⋮

图 3-2　标准化考试阅卷程序输入数据形式

标题记录只有一个，记录内容为统计报告的名字。

标准答案记录有若干个，其数目取决于试题数。标准答案记录的第 80 个字符均为"2"；每个记录的第 1～3 个字符表示试题数；第 10～59 个字符为标准答案，其中每个字符对应于一道试题的答案（标准答案记录中的第 1 个记录的第 10～59 个字符为第 1～50 题的答案，第 2 个记录的第 10～59 个字符为第 51～100 题的答案，以此类推）。

学生答卷记录也有若干个，其数目取决于学生人数和试题数。学生答卷记录的第 80 个字符均为"3"；每个记录的第 1～9 个字符表示学号；第 10～59 个字符为学生的答案，其中每个字

符对应于一道试题的学生答案(学生答卷记录中的第 1 个记录的第 10～59 个字符为第 1～50 题的答案,第 2 个记录的第 10～59 个字符为第 51～100 题的答案,以此类推)。

此外,规定学生数不大于 200,试题数不大于 999。

程序输出的统计报告有如下 4 种:

(1) 按学号排列的成绩单。按学号顺序列出每个学生的名次和成绩。

(2) 按学生成绩排列的成绩单,其中成绩相同者名次相同。

(3) 平均分和标准偏差报告。

(4) 试题分析报告。按试题号顺序列出每题学生回答正确的百分比。

下面首先采用边界值分析法设计测试用例,分别针对输入条件和输出条件的边界进行设计。针对输入条件边界可以设计若干测试用例,如表 3-4 所示。

表 3-4 阅卷程序针对输入条件边界设计的测试用例

测试用例序号	输 入 条 件	测 试 用 例
1	输入文件	输入文件为空
2～4	标题	没有标题记录 标题仅有一个字符 标题有 80 个字符
5～9	试题数	试题数为 1 试题数为 50 试题数为 51 试题数为 100 试题数为 0
10～12	标准答案记录	没有标准答案记录,有标题记录 标准答案记录多 1 个 标准答案记录少 1 个
13～16	学生人数	学生人数为 0 学生人数为 1 学生人数为 200 学生人数为 201
17	学生答卷	某学生答卷记录条数少于标准答案记录条数
18～19	学号	某学生学号为最小 某学生学号为最大
20～23	学生成绩	所有学生成绩相等 学生成绩均不相同 某一个学生成绩为 0 分 某一个学生成绩为 100 分

然后,针对输出条件(即输出的 4 个统计报告)边界设计若干测试用例,如表 3-5 所示。

表 3-5　阅卷程序针对输出条件边界设计的测试用例

测试用例序号	输 出 条 件	测 试 用 例
24～25	第 1、2 种报告	适当的学生人数，使统计报告刚好打印满 1 页 学生人数比刚才多 1
26～28	第 3 种报告	所有学生成绩均为 100 分 所有学生成绩均为 0 分 一半学生的成绩为 100 分，一半学生的成绩为 0 分(标准偏差最大)
29～34	第 4 种报告	所有学生均答对第 1 题 所有学生均答错第 1 题 所有学生均答对最后一题 所有学生均答错最后一题 适当的试题数，使统计报告刚好打印满 1 页 试题数比刚才多 1

通过执行表 3-4 和表 3-5 中各测试用例，可以发现该阅卷程序的大多数错误。

最后，进一步分析程序的规格说明，用错误推测法补充一些设计用例，如表 3-6 所示。

表 3-6　阅卷程序采用错误推测法补充的测试用例

测试用例序号	测 试 用 例
35	学生答卷记录的答案部分(第 10～59 个字符)含空格(不考虑每一学生的最后一条答卷记录)
36	除标题记录外，某些记录的第 80 个字符既不是“2”也不是“3”
37	有两个学生的学号相同
38	某学号中含有非数字字符
39	某学生的成绩是负值
40	部分学生成绩相同
41	试题数是负值

3.3.4　因果图法

1. 因果图法的实施策略

前面介绍的等价类划分法、边界值分析法及错误推测法在，对输入条件的考虑中并未重视输入条件的组合。事实上，当输入存在若干种可能的组合时，必须对这些组合加以考虑，以证实测试程序在某种输入组合的情况下能否完成规格说明书中预先规定的功能，否则所做的测试是不充分的。

因果图(Cause-effect Graph)是一种描述输入条件的组合及每种组合对应的输出的图形化工具。在因果图的基础上可以设计测试用例。

利用因果图设计测试用例应遵循如下步骤：

① 分析程序的规格说明中哪些是原因，哪些是结果。所谓原因，是指输入条件或输

入条件的等价类，而结果是指输出条件。给每个原因和结果赋一个标识符。

② 分析程序的规格说明中的语义，确定原因与原因、原因与结果之间的关系，画出因果图。

③ 由于语法或环境的限制，一些原因与原因之间、原因与结果之间的组合不能出现。对于这些特殊情况，在因果图中用一些记号标明约束或限制条件。

④ 将因果图转化为判定表。

⑤ 根据判定表的每一列设计测试用例。

当然，若能直接得到判定表，可直接根据判定表设计测试用例。

2. 因果图的画法

在因果图中，通常用 C_i 表示原因，用 E_i 表示结果，其基本符号如图 3-3 所示。

在因果图中，各结点的取值可为 0 或 1，0 表示该结点所代表的状态不出现，1 表示该结点代表的状态出现。对图 3-3 中的 4 种关系解释如下：

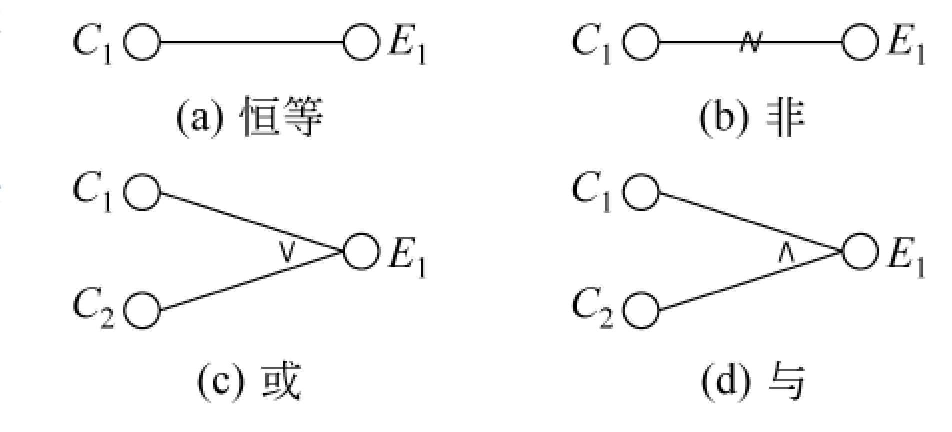


图 3-3 因果图的基本图形符号

(1) 恒等。若原因出现，则结果出现；若原因不出现，则结果不出现。

(2) 非。若原因出现，则结果不出现；若原因不出现，则结果出现。

(3) 或。若几个原因中有一个出现，则结果出现；若几个原因均不出现，结果才不出现。

(4) 与。若几个原因中都出现，结果才出现；若几个原因中有一个不出现，则结果不出现。

为了表示因果图中的约束条件，可用一些符号在因果图中加以标识。从原因方面考虑，主要有 4 种约束条件，如图 3-4(a)～(d)所示；从结果方面考虑，主要有 1 种约束条件，如图 3-4(e)所示。

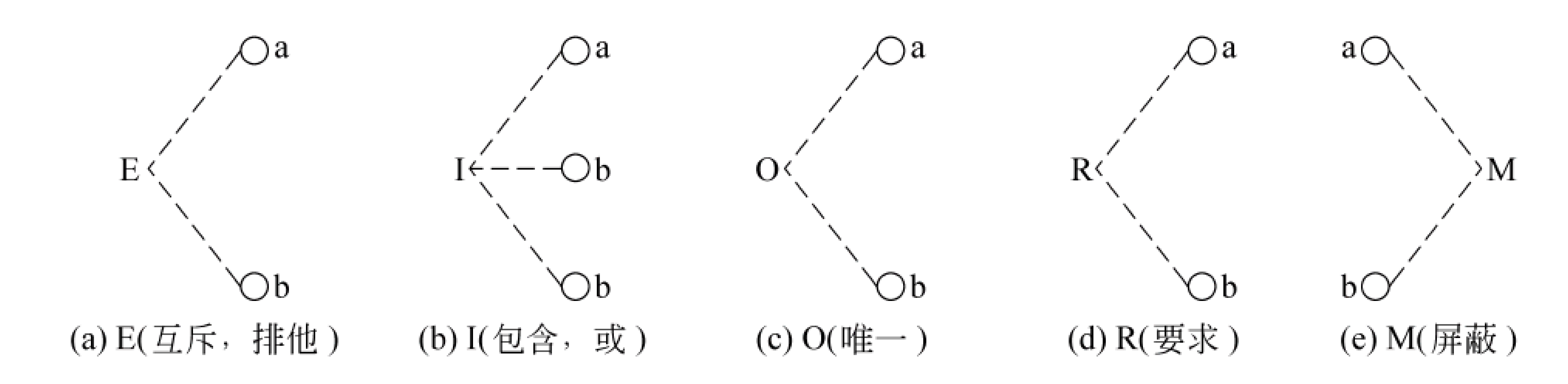


图 3-4 因果图中的约束符号

图 3-4 中的 5 种约束符号的含义分别如下：

(1) E(互斥)。a，b 两个原因不会同时出现，最多只有一个出现。

(2) I(包含)。a，b，c 3 个原因中至少有一个原因出现。

(3) O(唯一)。a，b 两个原因中必须有一个出现，且仅有一个出现。

(4) R(要求)。a 出现时 b 也必定出现。

(5) M(屏蔽)。a 出现时,b 必定不出现;a 不出现时,b 则不确定。

3. 因果图法举例

下面给出一个使用因果图法进行黑盒测试的范例。

有一个处理单价为 5 角钱饮料的自动售货机软件,对其采用因果图方法设计测试用例。该自动售货机软件规格说明中的要点如下:

① 若售货机没有零钱找,则一个显示“零钱找完”的红灯亮,以提示顾客在此情况下不要投入 1 元钱,否则此红灯不亮。

② 顾客投入 5 角硬币后,按下“橙汁”或“啤酒”按钮,则相应的饮料被送出。

③ 顾客投入 1 元硬币并按“橙汁”或“啤酒”按钮后,若售货机没有零钱找,则显示“零钱找完”的红灯亮,1 元硬币被退出,且无饮料送出;若有零钱找,则 5 角硬币被退出且饮料被送出。

(1) 首先在分析规格说明的基础上列出原因和结果。

原因有 5 种,如表 3-7 所示。

结果有 5 种,如表 3-8 所示。

表 3-7　自动售货机软件规格说明中的原因

编号	原　因	编号	原　因
1	售货机有零钱找	4	按“橙汁”按钮
2	投入 1 元硬币	5	按“啤酒”按钮
3	投入 5 角硬币		

表 3-8　自动售货机软件规格说明中的结果

编号	结　果	编号	结　果
21	售货机“零钱找完”灯亮	24	送出橙汁饮料
22	退还 1 元硬币	25	送出啤酒饮料
23	退还 5 角硬币		

(2) 通过对规格说明的进一步分析,在原因与结果之间再建立 4 个中间结点,如表 3-9 所示。

表 3-9　4 个中间结点

序　　号	中 间 结 点	序　　号	中 间 结 点
11	投入 1 元硬币且按饮料按钮	13	退还 5 角零钱且售货机有零钱找
12	按“橙汁”或“啤酒”按钮	14	钱已付清

(3) 根据列出的原因、结果和中间结点画出因果图,如图 3-5 所示。

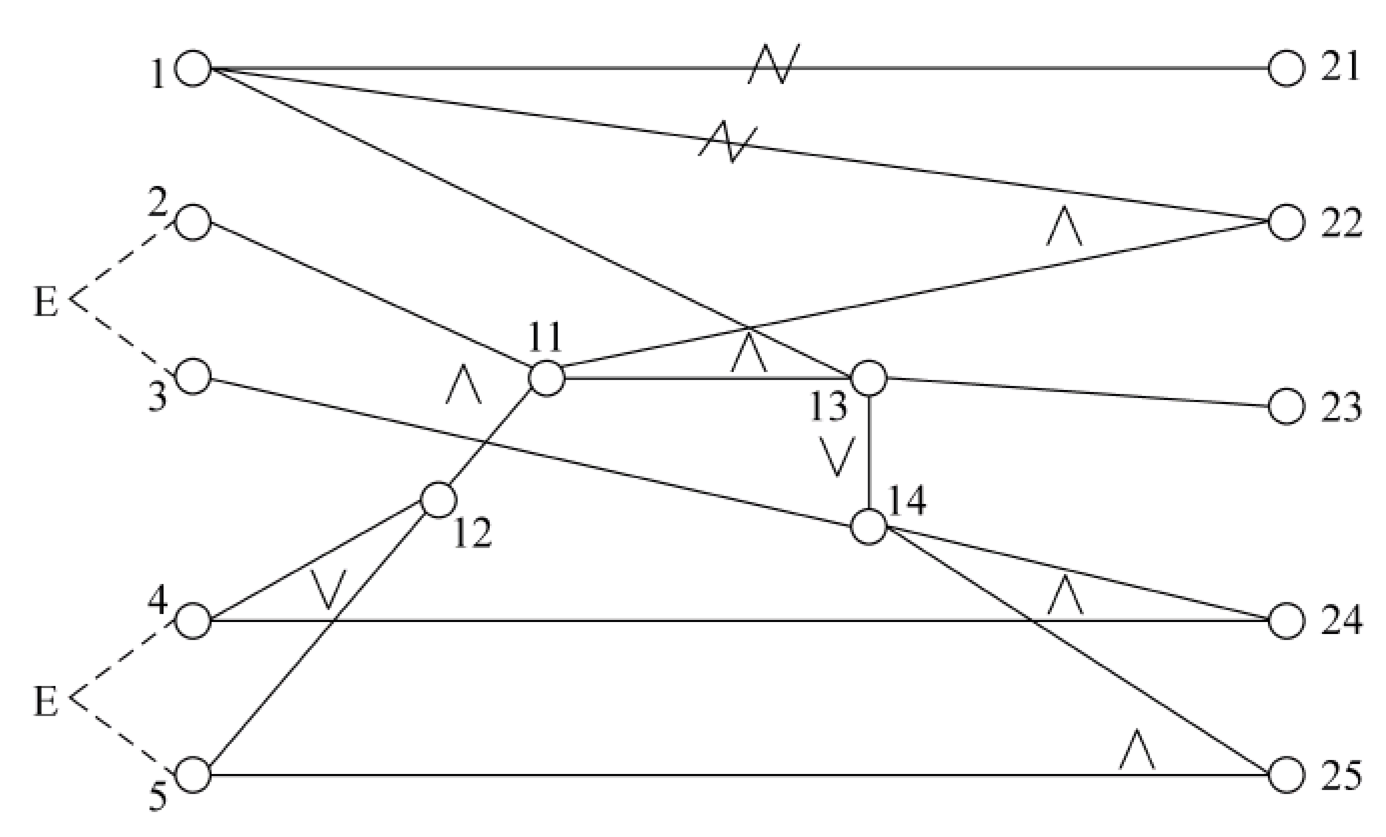


图 3-5 自动售货机软件的因果图

在图 3-5 中，原因在图的最左边，结果在图的最右边，中间结点则排列在图的中间位置。由于 2、3 号原因不能同时出现，4、5 号原因也不能同时出现，故在图中各用一个约束符号 E 加以标识。

(4) 将因果图转换为判定表，如表 3-10 所示。

在构成的判定表中，各原因、中间结点及结果的取值为 0 表示其代表的状态不出现；为 1 则表示状态出现。原因组合共有 32 列，但第 1～5，9，13，17～21，25，29 列因违反约束条件，第 16，32 列因没有进行任何操作，第 8，12，24，28 列因不符合常理(投币却没选择饮料)，均被视为无效的列。最后，根据 32 列中剩下的 12 列设计测试用例，可针对每一列设计一个测试用例。

这个例子中，事实上在分析问题时进行了一些简化，主要有以下几点：

① 判定表中第 8，12，24，28 列不符合常理(投币却没选饮料)，但却是可能发生的。可以考虑对规格说明做如下完善：若在投币后 n(如 10)秒后仍没选择饮料，自动售货机上“请选择饮料”的灯亮。进而根据规格说明改进因果图和判定表。

② 判定表中第 14，15，30，31 列表示只选择饮料却没投币。对规格说明应做如下完善：只选择饮料却没投币时，自动售货机上“请投币”的灯亮。进而根据规格说明改进因果图和判定表。

③ 判定表中第 26，27 列表示在自动售货机没零钱找的情况下投入了 5 角钱，此时自动售货机将会进入有零钱找的状态，由于在规格说明和因果图中没有考虑这一特殊情况，故判定表第 26，27 列的第 21 号结果没有从 1 变为 0，实际上是不正确的。在规格说明中应对此种情况加以考虑，进而改进因果图和判定表。

④ 判定表中第 6，7 列对应的是自动售货机在有零钱找的情况下找了 5 角钱，此时自动售货机可能进入无零钱找的状态，但在规格说明和因果图中没有考虑这一特殊情况，故判定表第 6，7 列的第 21 号结果的值不一定正确。在规格说明中应对此种情况加以考虑，进而改进因果图和判定表。

表 3-10　由自动售货机软件因果图得到的判定表

序　号		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
原因	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	2	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
	3	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
	4	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
	5	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
中间结点	11						1	1			0	0			0	0							1	1			0	0			0	0	
	12						1	1			1	1			1	1							1	1			1	1			1	1	
	13						1	1			0	0			0	0							0	0			0	0			0	0	
	14						1	1			1	1			0	0							0	0			1	1			0	0	
结果	21						0	0			0	0			0	0							1	1			1	1			1	1	
	22						0	0			0	0			0	0							1	1			0	0			0	0	
	23						1	1			0	0			0	0							0	0			0	0			0	0	
	24						1	0			1	0			0	0							0	0			1	0			0	0	
	25						0	1			0	1			0	0							0	0			0	1			0	0	

3.3.5 判定表法

在因果图方法中用到了判定表,判定表(Decision Table)也称为决策表,是软件工程实践中的重要工具,主要用在软件开发的详细设计阶段。判定表的作用与因果图类似,能表示输入条件的组合,以及与每一输入组合相对应的动作组合,因此判定表与因果图的使用场合类似。

判定表的构造形式如图 3-6 所示。

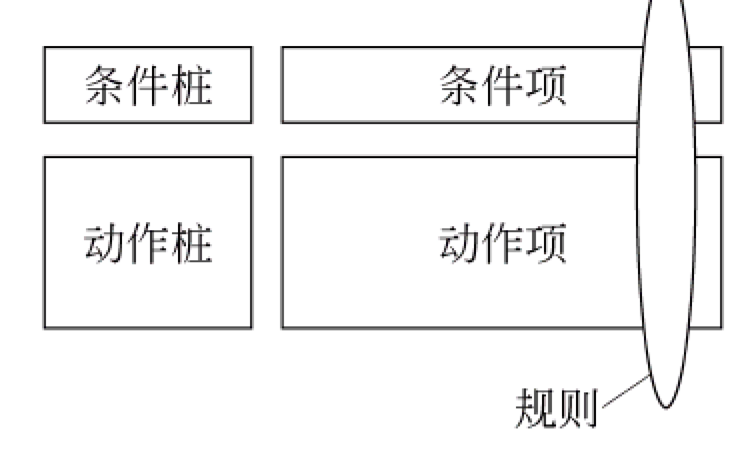


图 3-6 判定表形式

对判定表中的 5 个部分解释如下:

(1) 条件桩。列出所有可能的条件。条件的顺序一般不重要。

(2) 条件项。列出所有的条件取值组合。若有若干个条件项,每一条件项为一个条件取值组合。

(3) 动作桩。列出所有可能的操作。

(4) 动作项。列出在每一种条件取值组合情况下,执行动作桩中的哪些动作。故动作项的数目与条件项相等。

(5) 规则。一种条件取值组合与其对应的动作组合(即判定表中贯穿条件项和动作项的一列)构成判定表中的一个规则。条件取值组合的数目就是规则的数目,如在图 3-5 中,有 5 个原因,每个原因可取 0 或 1,故条件取值组合的数目为 32,所以规则也为 32 个。有一些规则因条件取值组合违反约束条件或不做任何动作而成为无效规则,可以废弃掉。

建立判定表可遵循的步骤如下:

① 列出条件桩和动作桩。

② 确定规则的个数,用来为规则编号。若有 n 个原因,由于每个原因可取 0 或 1,故有 2^n 个规则。

③ 完成所有条件项的填写。

④ 完成所有动作项的填写。

⑤ 合并相似规则,用以对初始判定表进行简化。

建立了判定表后,可针对判定表中的每一列有效规则设计一个测试用例,用于对程序进行黑盒测试。

3.3.4 小节和 3.3.5 小节介绍的因果图法和判定表法无论从使用场合还是实施策略来看都是十分类似的。一般说来,当输入条件存在多个取值组合,不同的取值组合对应于不完全相同的动作组合时,可考虑使用因果图法或判定表法。若能直接根据规格说明建立判定表,可用判定表法;若规格说明比较复杂,可先画出因果图,再建立判定表,利用判定表设计测试用例。关于判定法的实例可参见 3.3.4 小节的实例。

3.3.6 正交试验法

用黑盒方法设计测试用例时,由于输入条件和输出数据的多样性,欲使设计测试用例覆盖每一输入及输出取值组合成为一件十分困难的事,如何有效地缩减测试用例的数目,用较少的测试用例对软件进行较全面的测试,从而提高测试效率、控制测试成本?通过正交试验设计法进行测试用例的设计可以较好地达到这一目的。

为了能够清楚地理解对正交试验法设计测试用例，先看如下示例。

为提高某化学产品的转化率，选择 3 个有关因素进行条件试验，它们分别是反应温度 A、反应时间 B 和用碱量 C，并确定其取值范围为：

A：80～90℃；

B：90～150min；

C：5%～7%。

试验目的是确定因素 A,B,C 对转化率有什么影响，哪些是主要的，哪些是次要的，从而确定最佳生产条件，即使转化率最高时的温度、时间和用碱量。

注意如下两个概念：

- 因素：也称为因子，即试验中要考查的变量；
- 水平：在试验范围内变量的取值。

这里，对因子 A 在试验范围内选取了 3 个水平，因子 B 和 C 也都选取 3 个水平。

A：$A_1=80℃$，$A_2=85℃$，$A_3=90℃$；

B：$B_1=90\text{min}$，$B_2=120\text{min}$，$B_3=150\text{min}$；

C：$C_1=5\%$，$C_2=6\%$，$C_3=7\%$。

这个 3 因子 3 水平的条件试验，通常有下面两种试验方法。

（1）全面试验法

取 3 因子所有水平之间的组合，即 $A_1B_1C_1$，$A_1B_1C_2$，$A_1B_1C_3$，…，$A_3B_3C_3$，共 $3^3=27$ 个试验点（如图 3-7 所示立方体中的 27 个节点），故共需进行 27 次试验。

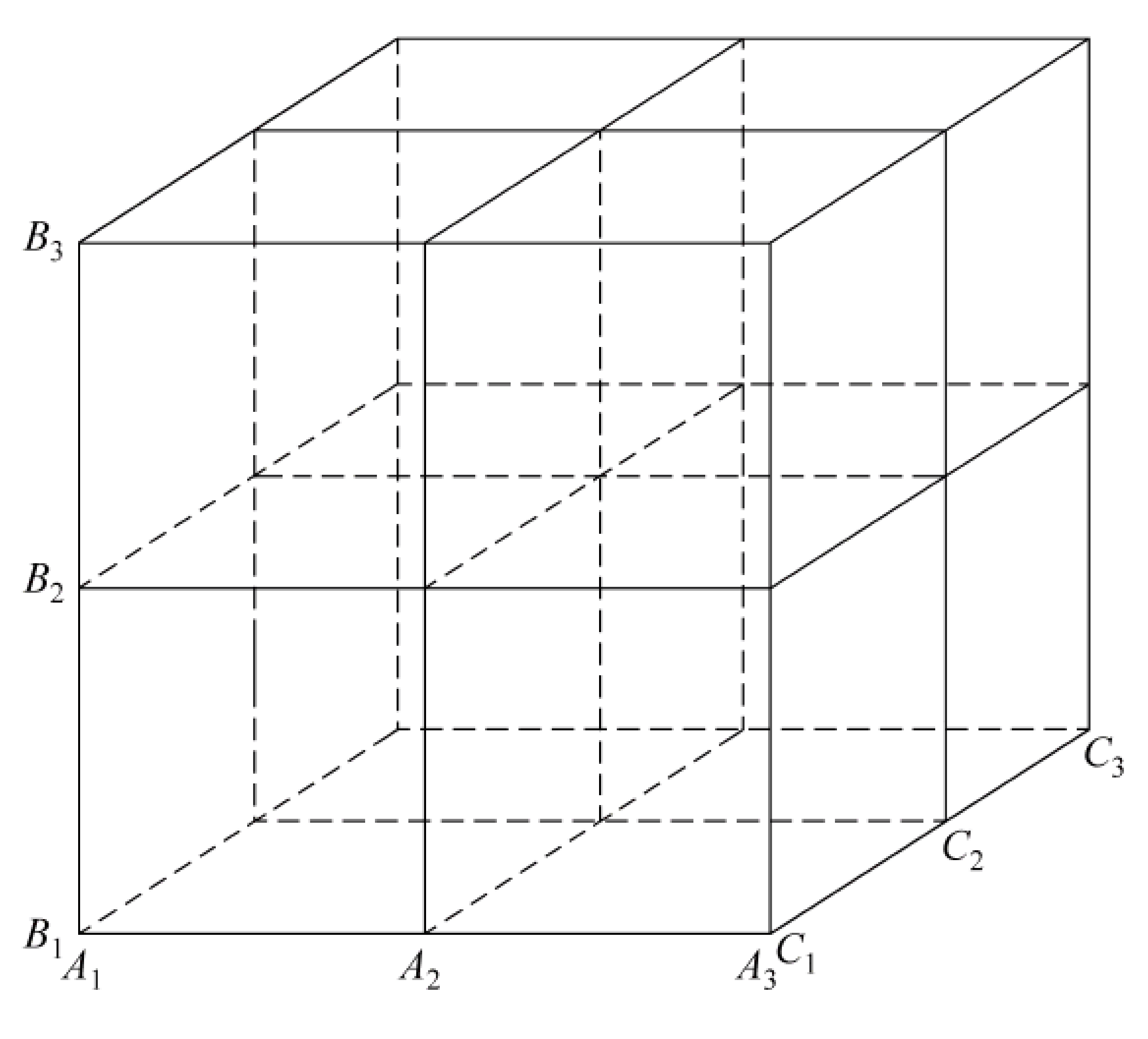


图 3-7　全面试验法图例

全面试验法无疑是一种对各种试验条件考虑最全面的方法，但当因子数和因子水平数增加时，试验次数急剧增加。所以，全面试验法只适合于因子数和因子水平数均很小的试验场合。

（2）单因素轮换试验法

单因素轮换试验是将多因素试验问题转化为单因素试验问题的处理方法。在这种试验

中，每次只变化一个因素，其他因素固定。例如对于上例，首先固定 B，C 于 B_1，C_1，使 A 变化：

$$B_1C_1 \to \begin{cases} A_1 \\ A_2 \\ A_3\ (\text{最好}) \end{cases}$$

若 A 取 A_3 时试验结果最好，则固定 A 于 A_3，C 仍取 C_1，使 B 变化：

$$A_3C_1 \to \begin{cases} B_1 \\ B_2\ (\text{最好}) \\ B_3 \end{cases}$$

若 B 取 B_2 时试验结果最好，则固定 B 于 B_2，A 于 A_3，使 C 变化：

$$A_3B_2 \to \begin{cases} C_1 \\ C_2\ (\text{最好}) \\ C_3 \end{cases}$$

若 C 取 C_2 时试验结果最好，就认为最好的工艺条件是 $A_3B_2C_2$。

单因素轮换试验法的最大优点就是试验次数少，但主要存在如下缺点：

- 试验点的分布并不均匀，因而试验不能全面地反应各种情况。例如按上述方法进行试验，试验点完全分布在一个角上，而没有在一个很大的范围内选点，因此所选的工艺条件 $A_3B_2C_2$ 不一定是 27 种组合中最好的。
- 用此方法比较试验条件时，是将单个试验数据进行数值上的简单比较，而试验数据中必然包含误差成分，因而会造成试验结论的不稳定。

如何兼具以上两种试验方法的优点，从全面试验的点中选择具有典型性、代表性的点，使试验点在试验范围内分布均匀，且试验点尽量地少呢？

例如上面的例子，对应于 A 有 A_1，A_2，A_3 3 个平面，对应于 B，C 也各有 3 个平面，共 9 个平面，则这 9 个平面上的试验点都应当一样多，即对每个因子的每个水平都要同等看待。具体来说，每个平面上都有 3 行、3 列，要求在每行、每列上的点一样多。这样，做出设计如图 3-8 所示，试验点用⊙表示。可以看到，在 9 个平面中每个平面上都恰好有 3 个

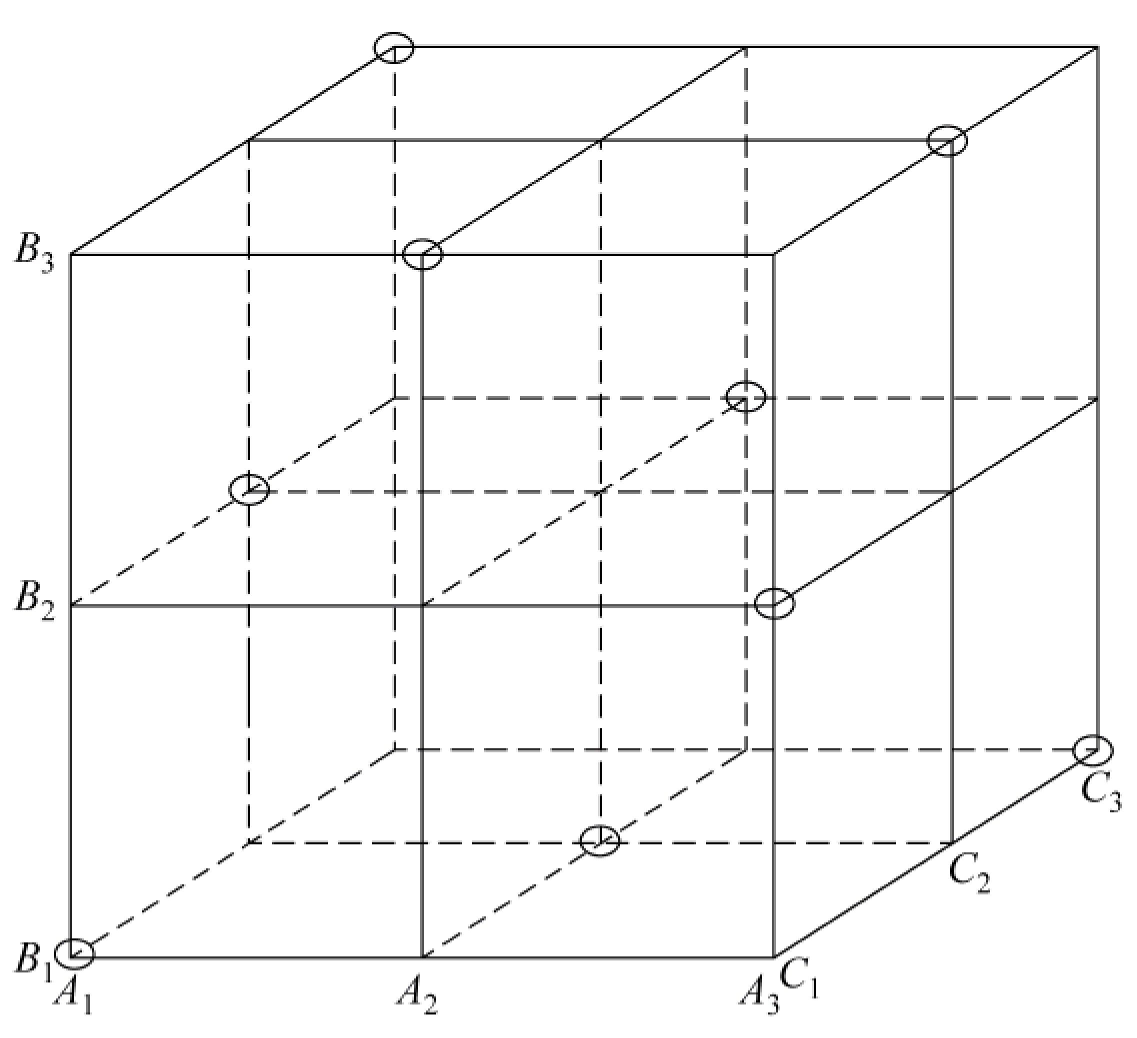


图 3-8　试验点均匀分布方案图例

点，而每个平面的每行每列都有 1 个点，且只有 1 个点，总共 9 个点。这样的试验方案，试验点的分布很均匀，试验次数也不多。

当因子数和水平数都不太大时，尚可通过作图的办法选择分布很均匀的试验点。但当因子数和水平数增加后，因子取值组合的数目会急剧增加，作图的方法就难以奏效了。此时可以采用正交试验设计方法。

正交试验设计方法是一种多因素多水平的试验设计方法。它应用依据 Galois 理论导出的正交表，从大量试验条件中挑选出适量的、有代表性的条件（即试验点）来合理地安排试验。运用这种方法安排的试验具有“均匀分散、整齐可比”的特点。均匀分散性使试验点均衡地分布在试验范围内，每个试验点有充分的代表性；整齐可比性使试验结果的分析十分方便，可以估计各因素对指标的影响，找出影响事物变化的主要因素。

正交表的形式为 $L_l(m^n)$，其中 l 为正交表的行数，即试验次数；n 为正交表列的数目，即最多可以安排的因子数；m 为因子的水平数。

常用的正交表有 $L_8(2^7)$，$L_9(3^4)$，$L_{16}(4^5)$，$L_8(4\times2^4)$，$L_{18}(2\times3^7)$等。例如 $L_8(2^7)$中，7 为正交表的列数，2 为因子的水平数，8 为正交表的行数即试验次数。又例如 $L_8(4\times2^3)$则表示，正交表中有 4 列是 2 水平的，1 列是 4 水平的，正交表共 8 行。

在行数为 ab（a，b 为正整数）的正交表中，行数$=\sum$（每列水平数-1）$+1$。利用该式可以通过所要考查的因子水平数决定最少的试验次数，从而选择合适的正交表。如要考查 5 个 3 水平因子和 1 个 2 水平因子，则最少的试验次数为 $5\times(3-1)+1\times(2-1)+1=12$，故应在行数不小于 12 且有 5 个以上 3 水平列和 1 个以上 2 水平列的正交表中进行选择，则以上给出的常用正交表中的 $L_{18}(2\times3^7)$适合。

正交表具有如下两条性质：

- 每一列中各数字出现的次数一样多。
- 任意两列中包含的各有序数对出现的次数一样多。

如在表 3-11 所示的 $L_9(3^4)$中，各列中的水平取值 1，2，3 都各出现 3 次，任意两列都包含 9 种数对，每种数对都各出现 1 次。

表 3-11　正交表 $L_9(3^4)$

行号＼列号	1	2	3	4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

下面我们回到本小节最开始的例子，看如何用正交试验设计方法为该例设计试验方案。显然，该例有 3 个 3 水平因子，故需进行的试验次数为 $3\times(3-1)+1=7$，故选取正交表 $L_9(3^4)$。正交表 $L_9(3^4)$有 4 列，而该例只有 3 个因子，每个因子对应正交表的一列，可仅使用表中前 3 列。设正交表 $L_9(3^4)$第 1 列水平取值 1，2，3 分别代表 $A_1=80℃$，$A_2=85℃$，$A_3=90℃$；第 2 列水平取值 1，2，3 分别代表 $B_1=90\text{min}$，$B_2=120\text{min}$，$B_3=150\text{min}$；第 3 列水平取值 1，2，3 分别代表 $C_1=5\%$，$C_2=6\%$，$C_3=7\%$。由此，可得出正交试验设计方案，如表 3-12 所示。

表 3-12 由 $L_9(3^4)$得出的正交试验方案

试 验 号	水 平 组 合	试 验 条 件		
		反应温度/℃	反应时间/min	用碱量/%
1	$A_1\ B_1\ C_1$	80	90	5
2	$A_1\ B_2\ C_2$	80	120	6
3	$A_1\ B_3\ C_3$	80	150	7
4	$A_2\ B_1\ C_2$	85	90	6
5	$A_2\ B_2\ C_3$	85	120	7
6	$A_2\ B_3\ C_1$	85	150	5
7	$A_3\ B_1\ C_3$	90	90	7
8	$A_3\ B_2\ C_1$	90	120	5
9	$A_3\ B_3\ C_2$	90	150	6

在表 3-11 中，每一行的各水平组合就构成了一个试验条件，也对应于测试中的一个测试用例。

由前文可知，对于此例，若用全面试验法需进行 27 次试验，而使用正交试验设计法，将试验次数缩减到 9 次，且这 9 次试验在一定意义上代表了 27 次试验，从而大大提高了试验的效率。

用正交试验设计方法设计测试用例的基本步骤如下：

(1) 确定因子。即确定对软件的运行结果有影响的因素。一般情况下是指软件的输入，以及软件运行的其他环境。这些因子可以通过对软件需求规格说明书进行分析而获得。若因子太多，可根据测试成本要求剔除对运行结果影响不大的因素。

(2) 确定因子的取值范围。这些数据可以通过对软件需求规格说明书进行分析而得到。

(3) 确定每个因子的水平。根据因子的取值范围，采用等价类划分法、边界值分析法及其他测试用例设计方法，在每个因子的取值范围内挑选出有代表性的值。

(4) 选择合适的正交表。根据确定的因子和水平，选择一张合适的正交表，以满足试验中的因子数、各因子水平数及试验次数的要求。

(5) 设计测试用例表。正交表中的一行对应于各因子的取值组合即水平组合，也就

是一个试验条件。故可根据正交表得出所有试验条件,从而得到测试用例表。若需要对某种测试结果进行直观分析,该表还要包括其他内容,如响应时间、响应时间之和、平均响应时间、极差等,以便于找出影响该结果的主次要因素。

对正交试验设计及正交表更深入的了解可参见概率论和数理统计方面的书籍。

3.3.7　场景法

现在的软件几乎都是用事件触发来控制流程的,事件触发时的情景即为场景。而同一事件不同的触发顺序和处理结果就形成了事件流。运用在软件设计中的场景法也可用在软件测试中,有利于测试人员设计测试用例,并使测试用例更容易理解和执行。

提出将场景法用于测试的是 IBM Rational 公司(Rational 公司已于 2003 年被 IBM 收购),在其 RUP2000 中文版中对场景法有详细的介绍。

以下根据 RUP2000 对场景法进行介绍。

1. 场景法的实施策略

场景法最终生成的测试用例来源于测试目标的用例。也就是说,场景法是从用例生成测试用例。

场景法通过用例场景描述业务操作流程。它为每个用例场景编制测试用例。用例场景要通过描述流经用例的路径来确定,这个流经过程从用例开始到结束遍历其中所有基本流(基本事件)和备选流(分支事件)。

例如,图 3-9 中经过用例的每条不同路径都反映了基本流和备选流,都用箭头来表示。基本流用直黑线表示,是经过用例的最简单的路径。每个备选流自基本流开始,之后备选流会在某个特定条件下执行。备选流可能会重新加入到基本流中(备选流 1 和 3),还可能起源于另一个备选流(备选流 2),或者终止用例而不再重新加入某个流(备选流 2 和 4)。

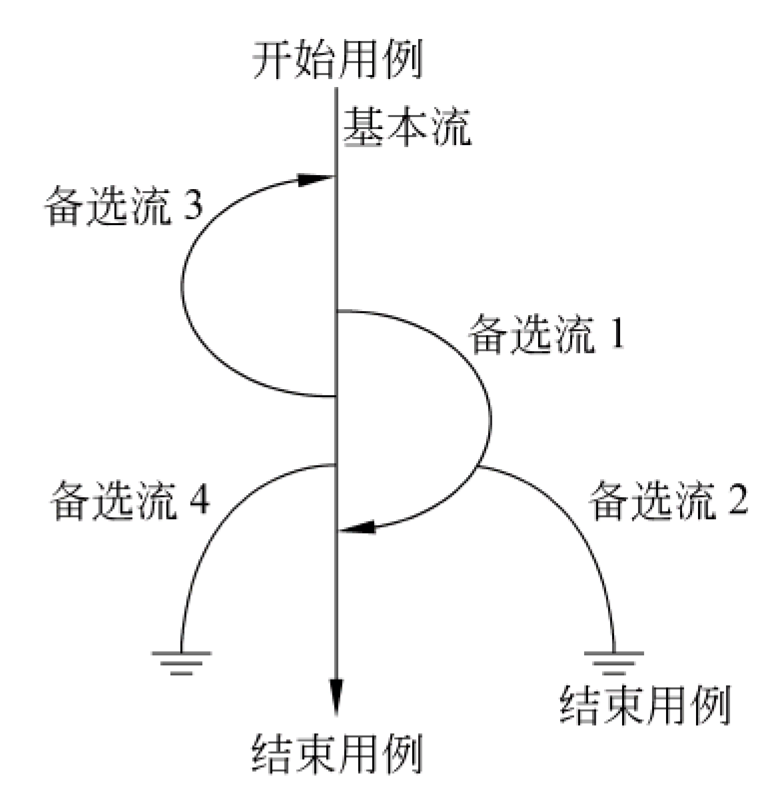


图 3-9　用例的事件流示例

遵循图 3-9 中每个经过用例的可能路径,可以确定不同的用例场景。从基本流开始,再将基本流和备选流结合起来,可以确定以下用例场景,如表 3-13 所示。

表 3-13　由图 3-9 确定的用例场景

场景 1	基本流			
场景 2	基本流	备选流 1		
场景 3	基本流	备选流 1	备选流 2	
场景 4	基本流	备选流 3		
场景 5	基本流	备选流 3	备选流 1	
场景 6	基本流	备选流 3	备选流 1	备选流 2
场景 7	基本流	备选流 4		
场景 8	基本流	备选流 3	备选流 4	

注意，为方便起见，场景 5、6 和 8 只描述了备选流 3 指示的循环执行一次的情况。生成每个场景的测试用例是通过确定某个特定条件来完成的，这个特定条件将导致特定用例场景的执行。

例如，假定图 3-9 描述的用例对备选流 3 规定如下：

如果在上述基本流步骤 2“输入提款金额”中输入的美元量超出当前账户余额，则出现此事件流。系统将显示一则警告消息，之后重新加入基本流，再次执行上述步骤 2“输入提款金额”，此时银行客户可以输入新的提款金额。

据此，可以开始确定需要用来执行备选流 3 的测试用例，如表 3-14 所示。

表 3-14　需要用来执行备选流 3 的测试用例

测试用例 ID	场景	条　件	预 期 结 果
TC x	场景 4	步骤 2—提款金额 > 账户余额	在步骤 2 处重新加入基本流
TC y	场景 4	步骤 2—提款金额 < 账户余额	不执行备选流 3，执行基本流
TC z	场景 4	步骤 2—提款金额 = 账户余额	不执行备选流 3，执行基本流

值得注意的是，由于没有提供其他信息，以上显示的测试用例都非常简单，实际测试中的测试用例很少如此简单。

2. 场景法举例

下面给出一个场景法(即从用例生成测试用例)的示例。

图 3-10 是 ATM 的流程图。

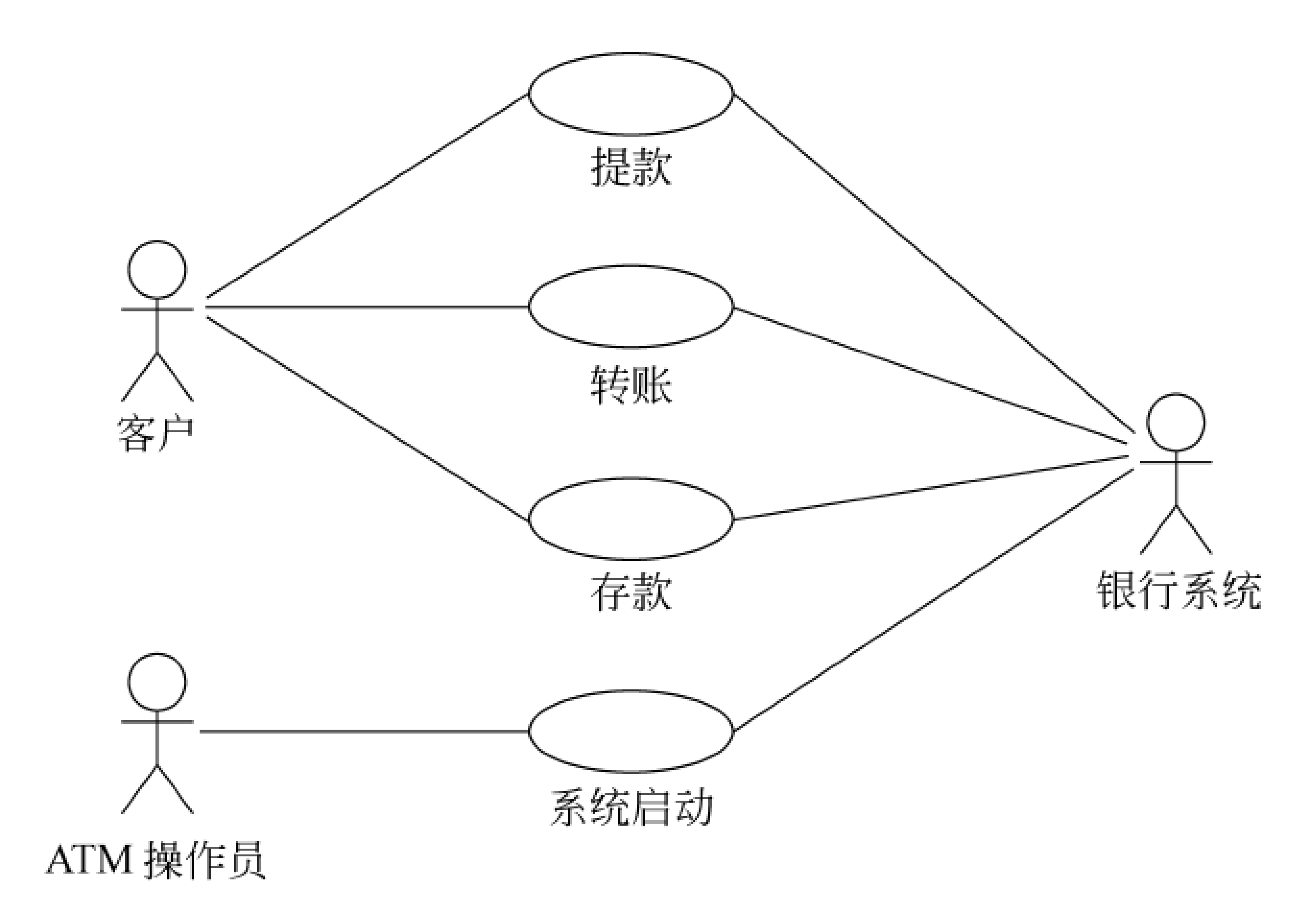


图 3-10　一台 ATM 机的主角和用例

表 3-15 包含了图 3-10 中提款用例的基本流和一些备用流。

表 3-15　图 3-8 中提款用例的基本流和某些备用流

基本流	本用例的开始是 ATM 处于准备就绪状态 ① 准备提款：客户将银行卡插入 ATM 机的读卡机 ② 验证银行卡：ATM 机从银行卡的磁条中读取账户代码，并检查它是否属于可以接收的银行卡 ③ 输入 PIN：ATM 机要求客户输入 PIN 码(4 位) ④ 验证账户代码和 PIN：验证账户代码和 PIN 以确定该账户是否有效，以及所输入的 PIN 对该账户来说是否正确；对于此事件流，账户是有效的，而且 PIN 对此账户来说正确无误 ⑤ ATM 机选项：ATM 机显示在本机上可用的各种选项。在此事件流中，银行客户通常选择"提款" ⑥ 输入金额：要从 ATM 机中提取的金额。对于此事件流，客户需选择预设的金额(10 美元，20 美元，50 美元或 100 美元) ⑦ 授权：ATM 机通过将卡 ID、PIN、金额及账户信息作为一笔交易发送给银行系统来启动验证过程。对于此事件流，银行系统处于联机状态，而且对授权请求给予答复，批准完成提款过程，并且据此更新账户余额 ⑧ 出钞：提供现金 ⑨ 返回银行卡：银行卡被返还 ⑩ 收据：打印收据并提供给客户，ATM 机还相应地更新内部记录 用例结束时，ATM 机又回到准备就绪状态
备选流 1：银行卡无效	在基本流步骤②验证银行卡中，如果卡是无效的，则卡被退回，同时会通知相关消息
备选流 2：ATM 机内没有现金	在基本流步骤⑤ATM 机选项中，如果 ATM 机内没有现金，则"提款"选项将无法使用
备选流 3：ATM 机内现金不足	在基本流步骤⑥输入金额中，如果 ATM 机内金额少于请求提取的金额，则将显示一适当的消息，并且在步骤⑥输入金额中，重新加入基本流
备选流 4：PIN 有误	① 在基本流步骤④中，验证账户和 PIN，客户有 3 次机会输入 PIN ② 如果 PIN 输入有误，ATM 机将显示适当的消息；如果还存在输入机会，则此事件流在步骤③输入 PIN 处重新加入基本流 ③ 如果最后一次尝试输入的 PIN 码仍然错误，则该卡将被 ATM 机保留，同时 ATM 机返回准备就绪状态，本用例终止
备选流 5：账户不存在	① 在基本流步骤④验证账户代码和 PIN 中，如果银行系统返回的代码表明找不到 ② 该账户或禁止从该账户中提款，则 ATM 机显示适当的消息并且在步骤⑨返回银行卡处重新加入基本流
备选流 6：账面金额不足	在基本流步骤⑦授权中，银行系统返回代码表明账户余额少于在基本流步骤⑥输入金额中输入的金额，则 ATM 机显示适当的消息并且在步骤⑥输入金额中重新加入基本流
备选流 7：达到每日最大的提款金额	在基本流步骤⑦授权中，银行系统返回的代码表明包括本提款请求在内，客户已经或将超过在 24 小时内允许提取的最多金额，则 ATM 机显示适当的消息并在步骤⑥输入金额上重新加入基本流
备选流 x：记录错误	如果在基本流步骤⑩收据中记录无法更新，则 ATM 机进入"安全模式"。在此模式下所有功能都将暂停使用，同时向银行系统发送一条适当的警报信息表明 ATM 机已经暂停工作

续表

备选流 y：退出	客户可随时决定终止交易(退出)。交易终止,银行卡随之退出
备选流 z："翘起"	ATM 机包含大量的传感器,用于监控各种功能,如电源检测器、不同的门和出入口处的测压器,以及动作检测器等。在任一时刻,如果某个传感器被激活,则警报信号将发送给警方且 ATM 机进入"安全模式",在此模式下所有功能都暂停使用,直到采取适当的重启/重新初始化的措施
在第一次迭代中,根据迭代计划,我们需要核实提款用例已经正确地实施。此时尚未实施整个用例,只实施了下面的事件流。 基本流：提取预设金额(10 美元,20 美元,50 美元,100 美元) 备选流 2：ATM 内没有现金 备选流 3：ATM 内现金不足 备选流 4：PIN 有误 备选流 5：账户不存在/账户类型有误 备选流 6：账户金额不足	

从这个用例可以生成下列场景,如表 3-16 所示。

表 3-16 场景设计

场景	描 述	基本流	备选流
场景 1	成功提款	基本流	
场景 2	ATM 内没有现金	基本流	备选流 2
场景 3	ATM 内现金不足	基本流	备选流 3
场景 4	PIN 有误(还有输入机会)	基本流	备选流 4
场景 5	PIN 有误(不再有输入机会)	基本流	备选流 4
场景 6	账户不存在/账户类型有误	基本流	备选流 5
场景 7	账户余额不足	基本流	备选流 6

注意,为了方便起见,备选流 3 和 6(场景 3 和 7)内的循环及循环组合未纳入表 3-16 中。

对于这 7 个场景中的每一个场景,都需要确定测试用例。可以采用矩阵或判定表来确定和管理测试用例。表 3-17 显示了一种通用格式,其中各行代表各个测试用例,而各列则代表测试用例的信息。本示例中,对于每个测试用例,存在一个测试用例 ID、条件(或说明)、测试用例中涉及的所有数据元素(作为输入或已经存在于数据库中)以及预期结果。

通过从确定执行用例场景所需的数据元素入手构建矩阵,然后对于每个场景至少要确定包含执行场景所需的适当条件的测试用例。例如,表 3-17 中,V(有效)用于表明这个条件必须是 Valid(有效的)才可执行基本流,而 I(无效)用于表明这种条件下将激活所需备选流;n/a(不适用)表明这个条件不适用于测试用例。

表 3-17　测试用例表

TC(测试用例)ID 号	场景/条件	PIN	账号	输入的金额(或选择)的金额	账面金额	ATM 机内的金额	预期结果
CW1	场景 1：成功提款	V	V	V	V	V	成功地提款
CW2	场景 2：ATM 内没有现金	V	V	V	V	I	提款选项不可用，用例结束
CW3	场景 3：ATM 内现金不足	V	V	V	V	I	警告消息，返回基本流步骤 ⑥输入金额
CW4	场景 4：PIN 有误（还有不止一次输入机会）	I	V	n/a	V	V	警告消息，返回基本流步骤③，输入 PIN
CW5	场景 4：PIN 有误（还有一次输入机会）	I	V	n/a	V	V	警告消息，返回基本流步骤③，输入 PIN
CW6	场景 4：PIN 有误（不再有输入机会）	I	V	n/a	V	V	警告消息，卡予保留，用例结束

表 3-17 中，6 个测试用例执行了 4 个场景。对于基本流，上述测试用例 CW1 称为正面测试用例。它一直沿着用例的基本流路径执行，未发生任何偏差。基本流的全面测试必须包括负面测试用例，以确保只有在符合条件的情况下才执行基本流。这些负面测试用例由 CW2～CW6 表示（阴影单元格表明这种条件下需要执行备选流）。虽然 CW2～CW6 对于基本流而言都是负面测试用例，但它们相对于备选流 2～4 而言是正面测试用例。而且对于这些备选流中的每一个而言，至少存在一个负面测试用例（CW1，基本流）。

每个场景只具有一个正面测试用例和负面测试用例是不充分的，场景 4 正是这样的一个示例。要全面地测试场景 4PIN 有误，至少需要下面 3 个正面测试用例（以激活场景 4）。

① 输入了错误的 PIN，但仍存在输入机会，此备选流重新加入基本流中的步骤③输入 PIN。

② 输入了错误的 PIN，而且不再有输入机会，则此备选流将保留银行卡并终止用例。

③ 最后一次输入时输入了正确的 PIN。备选流在步骤⑥输入金额处重新加入基本流。

注意，表 3-17 中无须为条件（数据）输入任何实际的值。以这种方式创建测试用例表（矩阵）的一个优点在于容易看到测试的条件。由于只需要查看 V 和 I（或此处采用的阴影单元格），这种方式还易于判断是否已经确定了充足的测试用例。从表中可发现存在几个条件不具备阴影单元格，表明测试用例还不完全，如场景 6 不存在的账户/账户类型有误和场景 7 账户余额不足就缺少测试用例。

一旦确定了所有测试用例，则应对这些用例进行复审和验证以确保其准确且适度，并取消多余或等效的测试用例。

测试用例一经认可，就可以确定实际数据值（在测试用例实施矩阵中）并且设定测试

数据，如表 3-18 所示。

表 3-18 测试数据表

TC(测试用例)ID 号	场景/条件	PIN	账号	输入的金额（或选择的金额）	账面金额	ATM 内的金额	预期结果
CW1	场景 1：成功提款	4987	809—498	50.00	500.00	2000.00	成功地提款。账户余额被更新为 450.00
CW2	场景 2：ATM 内没有现金	4987	809—498	100.00	500.00	0.00	提款选项不可用，用例结束
CW3	场景 3：ATM 内现金不足	4987	809—498	100.00	500.00	70.00	警告消息，返回基本流步骤⑥输入金额
CW4	场景 4：PIN 有误（还有不止一次输入机会）	4978	809—498	n/a	500.00	2000.00	警告消息，返回基本流步骤③输入 PIN
CW5	场景 4：PIN 有误（还有一次输入机会）	4978	809—498	n/a	500.00	2000.00	警告消息，返回基本流步骤③输入 PIN
CW6	场景 4：PIN 有误（不再有输入机会）	4978	809—498	n/a	500.00	2000.00	警告消息，卡予保留，用例结束

以上测试用例只是在本次迭代中需要用来验证提款用例的一部分测试用例，需要的其他测试用例如下。

场景 6 账户不存在/账户类型有误：未找到账户或账户不可用。

场景 6 账户不存在/账户类型有误：禁止从该账户中提款。

场景 7 账户余额不足：请求的金额超出账户金额。

在将来的迭代中，当实施其他事件流时，在下列情况下将需要测试用例：

- 无效卡(所持卡为挂失卡、被盗卡、非承兑银行发卡、卡磁条损坏等)。
- 无法读卡(读卡机堵塞、脱机或出现故障)。
- 账户已销户、冻结或由于其他方面原因无法使用。
- ATM 机内现金不足或不能提供所请求的金额。注意与 CW3 不同，在 CW3 中只是一种币值不足，而不是所有币值都不足。
- 无法联系银行系统以获得认可。
- 银行网络离线或交易过程中断电。

3.3.8 功能图法

每个程序的功能通常由静态说明和动态说明组成，动态说明描述了输入数据的次序或者转移的次序；静态说明描述了输入条件与输出条件之间的对应关系。对于比较复杂的程序，由于大量组合情况的存在，如果仅仅使用静态说明来组织测试往往是不够的，还

必须有动态说明来补充。功能图法就是因此而产生的一种测试用例设计方法。

功能图法就是使用功能图形式化地表示程序的功能说明，并机械地生成功能图的测试用例。功能图模型由状态迁移图和逻辑功能模型组成。

(1) 状态迁移图用于表示输入数据序列及相应的输出数据，由输入和当前的状态决定输出数据和后续状态。

(2) 逻辑功能模型用于表示在状态中输入条件与输出条件之间的对应关系。逻辑功能模型只适合于描述静态说明，输出数据仅仅由输入数据决定。测试用例由测试中经过的一系列状态，以及在每个状态中必须依靠输入/输出数据满足的一对条件组成。

功能图法是一种黑盒和白盒混合用例设计方法。在功能图法中，需要用到逻辑覆盖和路径测试的概念和方法，这属于白盒设计方法；而确定输入数据序列及相应的输出数据，则是黑盒设计方法。

3.4　黑盒测试方法的综合使用策略

在使用黑盒测试方法时，只有结合被测软件的特点，有选择地使用若干种方法，方能达到良好的测试效果。

黑盒测试方法的综合使用策略如下：

(1) 首先进行等价类划分，包括输入条件和输出条件的等价划分，将无限测试变成有限测试，这是减少工作量和提高测试效率最有效的方法。等价类划分也常是边界值方法的基础。

(2) 在任何情况下都必须使用边界值分析方法。经验表明，用这种方法设计出的测试用例发现程序错误的能力最强。

(3) 测试人员可以根据经验用错误推测法追加一些测试用例。

(4) 如果程序的功能说明中含有输入条件的组合情况，则一开始就可选用因果图法和判定表法。

(5) 对于参数配置类软件，应用正交试验法选择较少的组合方式以达到最佳效果，并减少测试用例的数目。

(6) 对于业务流清晰的系统可以利用场景法。即可先综合使用各种方法生成用例，再通过场景法由用例生成用例。

(7) 当程序的功能较复杂、存在大量组合情况时，可以考虑使用功能图法。

3.5　小结

本章介绍了测试用例的含义和设计准则。测试用例的设计是软件测试中的重中之重，直接关系到测试工作的成功与否。黑盒测试法在软件测试用例的设计方法中占有重要的地位，相对于白盒测试方法，黑盒测试方法是站在较宏观的角度设计测试用例和进行测试活动的。本章介绍的几种黑盒测试方法中，每种方法都有自身的特点，实用时需结合

具体的被测软件的特点灵活地选择黑盒测试方法。

习　题

1. 什么是测试用例？设计测试用例时应遵循哪些基本准则？
2. 测试用例的设计方法主要包括哪几种？
3. 黑盒测试的含义是什么？
4. 需求规格说明的全面、准确与否将影响黑盒测试的执行效果，为什么？
5. 边界值方法和等价类方法的关系是怎样的？
6. 错误推测法设计测试用例的依据是什么？
7. 有一个评定并打印学生成绩等级的程序，其规格说明如下：

成绩满分为100分，学生成绩记为 x。若 $90 \leqslant x \leqslant 100$，打印等级为“优”；若 $80 \leqslant x < 90$，打印等级为“良”；若 $70 \leqslant x < 80$，打印等级为“中”；若 $60 \leqslant x < 70$，打印等级为“及格”；若 $0 \leqslant x < 60$，打印等级为“不及格”；若 $x < 0$ 或 $x > 100$ 或 x 中含有非数字字符，打印为“无效成绩”。试根据此规格说明用等价类方法、边界值方法和错误推测法共同完成针对该程序功能的黑盒测试用例设计。

8. 在什么情况下应使用因果图法或判定表法？
9. 在3.3.4小节介绍的自动售货机软件的例子中，对其规格说明做如下完善：若在投币 n（如10）秒后仍没选择饮料，自动售货机上“请选择饮料”的灯亮，请根据规格说明改进因果图和判定表。
10. 表3-19和表3-20是对某IC卡加油机应用系统的基本流和备选流描述，试回答问题(1)～(3)。

表3-19　基本流

序号	用例名称	用例描述
A1	准备加油	客户将IC加油卡插入加油机
A2	验证加油卡	加油机从加油卡的磁条中读取账户代码，并检查它是否属于可以接收的加油卡
A3	验证黑名单	加油机验证卡账户是否存在于黑名单中，如果属于黑名单，加油机吞卡
A4	输入购油量	客户输入需要购买的汽油数量
A5	加油	加油机完成加油操作，从加油卡中扣除相应金额
A6	返回加油卡	退还加油卡

表 3-20 备选流

序号	用例名称	用例描述
B	加油卡无效	在基本流 A2 过程中，该卡不能够识别或是非本机可以使用的 IC 卡，加油机退卡，并退出基本流
C	卡账户属于黑名单	在基本流 A3 过程中，判断该卡账户属于黑名单。例如已经挂失，加油机吞卡退出基本流
D	加油卡账面现金不足	系统判断加油卡内现金不足，重新加入基本流 A4，或选择退卡
E	加油机油量不足	系统判断加油机内油量不足，重新加入基本流 A4，或选择退卡

(1) 使用场景法设计测试用例，指出场景涉及的基本流和备选流，基本流和备选流用表中描述的相应字母表示。

(2) 对每一个场景都需要确定测试用例。

根据表 3-21 所示的通用格式(可参见 3.3.7 小节对表 3-15 的介绍)，首先确定执行用例场景所需数据元素(本例中包括账号、是否黑名单卡、输入油量、账户金额、加油机油量)，然后构建测试用例表，最后确定包含执行场景所需的适当条件的测试用例。

在表 3-21 中，V 表示有效数据元素，I 表示无效数据元素，n/a 表示不适用。例如，C01 表示“成功加油”基本流。请按上述规定为其他应用场景设计测试用例。

表 3-21 测试用例表

测试用例 ID 号	场景	账号	是否黑名单卡	输入油量	账面金额	加油机油量	预期结果
C01	场景 1：成功加油	V	I	V	V	V	成功加油
C02							
C03							
C04							
C05							

(3) 假如每升油 4 元人民币，用户的账户金额为 1000 元，加油机内油量足够，那么在 A4 输入油量的过程中，请运用边界值分析方法为 A4 选取合适的输入数据(即油量，单位：升)。

第 4 章

白盒测试方法

本章要点：

- 白盒测试的含义。
- 白盒测试无法实现穷举测试的原因。
- 白盒测试的优点和局限性。
- 白盒测试中的静态测试和动态测试。
- 逻辑覆盖法、基本路径测试和控制结构测试。
- 白盒测试方法的综合使用策略。

在这一章中将介绍白盒测试方法的概念和特点，以及典型的白盒测试方法，主要包括代码检查、静态结构分析、代码质量度量、逻辑覆盖法、基本路径测试、控制结构测试、程序插桩等，最后介绍了白盒测试方法的综合使用策略，并对黑盒和白盒测试方法进行了最后的总结。

4.1　白盒测试方法概述

4.1.1　白盒测试的概念

白盒测试（White-box Testing）也称结构测试、逻辑驱动测试或基于程序本身的测试。“白盒”将程序形象地比喻为放在一个透明的盒子里，故测试人员是了解被测程序的内部结构的。测试人员利用程序的内部逻辑结构和相关信息，对程序的内部结构和路径进行测试，检验其是否达到了预期的设计要求。白盒测试是从程序设计者的角度进行的测试。

白盒测试方法和黑盒测试方法一样，也不能做到穷举测试。这是由为程序的结构往往是复杂的，程序中很难完全不出现选择结构和循环结构，当程序中出现了选择结构和循环结构，会使程序中的路径数目大大增加，如果选择结构自身、选择结构与循环结构、循环结构自身会再出现嵌套，路径数则更是急剧增加。所以白盒测试若要覆盖程序中如天文数字般的全部路径，花费的时间和代价几乎是所有的测试项目所不允许的。

因此，白盒测试是一项技术含量很高的工作，测试人员采用白盒测试方法设计测试用

例时，必须在仔细研究程序内部结构的基础上，从数量极大的可用测试用例中精心挑选尽可能少的测试用例，来覆盖程序的内部结构。

4.1.2　白盒测试的优点和局限性

1. 白盒测试的优点

与黑盒测试相比，白盒测试深入到程序的内部进行测试，更易于定位错误的原因和具体位置，弥补了黑盒测试只能从程序外部进行测试的不足。

2. 白盒测试的局限性

即使白盒测试覆盖了程序中的所有路径，仍不一定能发现程序中的全部错误，这是由白盒测试的局限性所致。

- 白盒测试不能查出程序中的设计缺陷。
- 白盒测试不能查出程序是否遗漏了功能或路径。
- 白盒测试可能发现不了一些与数据相关的错误。

4.1.3　白盒测试方法的分类

白盒测试分为静态测试（Static Testing）和动态测试（Dynamic Testing）两大类。

静态测试不需要实际执行程序，静态测试的主要目的是检查软件的表示和描述是否一致，是否存在冲突和歧义。静态测试可以由人工执行，也可以借助自动化工具完成。

动态测试需要实际运行测试用例，以发现软件中的错误。白盒测试中的动态测试主要包括功能确认与接口测试、覆盖率测试、性能分析、内存分析等。

（1）功能确认与接口测试

功能确认与接口测试主要用于测试各个单元能否完成指定的功能，以及单元间的接口是否正确。测试的对象包括局部数据结构、重要的执行路径、错误处理路径、单元接口和影响上述几点的边界条件等。

（2）覆盖率测试

覆盖率测试关注的是测试用例对程序中可执行路径的覆盖范围，常用的覆盖标准包括语句覆盖、判定覆盖、条件覆盖、判定—条件覆盖、条件组合覆盖、基本路径覆盖等。为判断测试用例对代码的覆盖是否达到指定的覆盖标准，一般还应在执行测试之后对覆盖率进行统计，进而针对未覆盖的部分补充测试用例。

（3）性能分析

代码运行是否缓慢是开发过程中需要注重的一个重要的性能问题。若不能解决此问题，将极大地影响应用程序的质量，应查找到影响程序性能的瓶颈并解决之。目前，性能分析工具大致分为纯软件的测试工具、纯硬件的测试工具（如逻辑分析仪和仿真器等）和软硬件结合的测试工具三类。

（4）内存分析

内存泄漏会导致系统运行的崩溃。通过测量内存的使用情况，可以了解程序内存分配的真实情况，发现对内存的不正常使用，在问题出现前发现征兆，并精确显示发生错误时的上下文情况。

4.2 典型的白盒测试方法

这一节将要介绍的白盒测试方法中的代码检查法、静态结构分析法和代码质量度量法属于静态测试方法，而逻辑覆盖法、基本路径测试法、控制结构测试和程序插桩则属于动态测试方法。

4.2.1 代码检查法

代码检查包括桌面检查、代码审查和走查等。代码检查主要检查代码和设计的一致性，代码是否遵循标准，代码的可读性，代码的逻辑正确性，代码结构的合理性等。代码检查可以发现违背程序编写标准的问题及程序中不安全、不明确和模糊的部分，找出程序中不可移植部分、违背程序编程风格的问题，包括变量检查、命名和类型审查、程序逻辑审查、程序语法检查和程序结构检查等内容。

代码检查所发现的是问题本身而非征兆。在实际使用中，代码检查是一种有效的测试手段，能发现30%～70%的逻辑设计和编码缺陷。但是代码检查也是一项较为费时的工作，而且代码检查需要编程及测试方面知识、经验的积累。

代码检查应在编译和动态测试之前进行，并且应尽早进行，若等到编码快结束或已结束时再进行代码检查，修复一个错误往往会引起一系列变更，付出较大的代价。在进行代码检查前，应准备好需求规格说明书、设计规格说明书、程序源代码清单、编码标准和代码缺陷检查表等资料；进行代码检查的人员应认真研读这些资料，以提高代码检查的效率。

下面分别介绍代码检查的3种形式：桌面检查、代码审查和走查。

1. 桌面检查

桌面检查(Desktop Checking)是一种传统的检查方法，由编程人员检查自己编写的程序。编程人员对自己编写的源代码进行分析、检验，甚至人工执行代码，以尽可能地发现程序中的错误。

由于编程人员熟悉自己的编程思路和程序逻辑结构，故桌面检查可以节省检查时间。但也正是因为检查自己的程序，有时会跳不出自己的思维框架，难以发现程序中的潜在问题，应注意防止这种倾向。

2. 代码审查

代码审查(Code Inspection)是由程序的编写者、其他编程人员和测试人员共同完成的。

在代码审查之前，审查小组负责人将设计规格说明书、程序清单及编码规范等分发给小组成员，作为审查的依据。特别是要给小组每个成员一份常见错误清单，也称为缺陷检查表，它罗列了以往编程中的常见错误，并对错误进行了分类。

审查小组成员在仔细研读以上材料后，进入正式的审查过程，即召开代码审查会。在会上，编程人员对自己编写的程序，逐句讲解其中的逻辑，审查小组其他成员则可以提出各自的疑问，进而展开讨论，以确认错误是否存在。

大量实践表明，编程人员在讲解自编程序的过程中，更易于发现原先未发现的问题，

而小组所有成员的共同讨论,也有利于错误的暴露,甚至发现一系列连带问题,从而大大提高软件的质量。

3. 走查

走查(Walkthrough)与代码审查的步骤基本相同。在进行走查之前,走查小组的成员也会得到设计规格说明书、程序清单、编码规范等与被测程序相关的资料,走查小组的成员应充分研读这些资料。在走查的过程中,与代码审查不同的是,小组成员不再简单地逐一分析程序的逻辑,而是由走查小组中的测试人员设计一批有代表性的测试用例,走查小组的成员则集体扮演计算机的角色,在头脑中沿着程序的逻辑运行这些测试用例,将程序运行的踪迹记录于纸上或黑板上供大家分析和讨论,进而检查其执行逻辑、控制模型、算法和使用参数与数据的正确性,以发现设计中存在的问题,以及设计与编码不一致之处。

4.2.2　静态结构分析法

静态结构分析主要是以图形的方式表现程序的内部结构,供测试人员对程序结构进行分析。

在静态结构分析中,测试人员通过使用测试工具分析程序源代码的系统结构、数据结构、数据接口、内部控制逻辑等内部结构,生成函数调用关系图、模块控制流图、内部文件调用关系图、子程序表、宏和函数参数表等各类图形图表,以清晰地表示程序的内部结构,供测试人员对其进行分析,进而查找程序中的错误。

静态结构分析法在此不加详述,读者可参阅其他资料。

4.2.3　代码质量度量法

根据 ISO/IEC 9126 国际标准的定义,软件质量包括以下 6 个方面:

- 功能性(functionality);
- 可靠性(reliability);
- 可用性 (usability);
- 效率 (efficiency);
- 可维护性(maintainability);
- 可移植性(portability)。

根据 ISO 9126 质量模型,可以构造软件的静态质量度量模型,通过量化的数据评估被测程序的质量。

4.2.4　逻辑覆盖法

逻辑覆盖法(Logic-coverage Testing)是以程序内部的逻辑结构为基础设计测试用例的方法。根据对程序内部逻辑结构的覆盖程度,逻辑覆盖法具有不同的覆盖标准:语句覆盖、判定覆盖、条件覆盖、判定—条件覆盖、条件组合覆盖和修正条件判定覆盖。

1. 语句覆盖

语句覆盖(Statement Coverage)的含义是,设计足够多的测试用例,使被测程序中的每条可执行语句至少执行一次。语句覆盖也称为点覆盖。

图 4-1 是一个被测程序的流程图。

对图 4-1 表示的程序，若要做到语句覆盖，程序的执行路径应是 sacbed，为此可设计如下的测试用例（注意：A，B，X 的值这里为输入值，严格说来，测试用例还应包括预期输出，此处省略，下同）：

A＝2， B＝0， X＝4

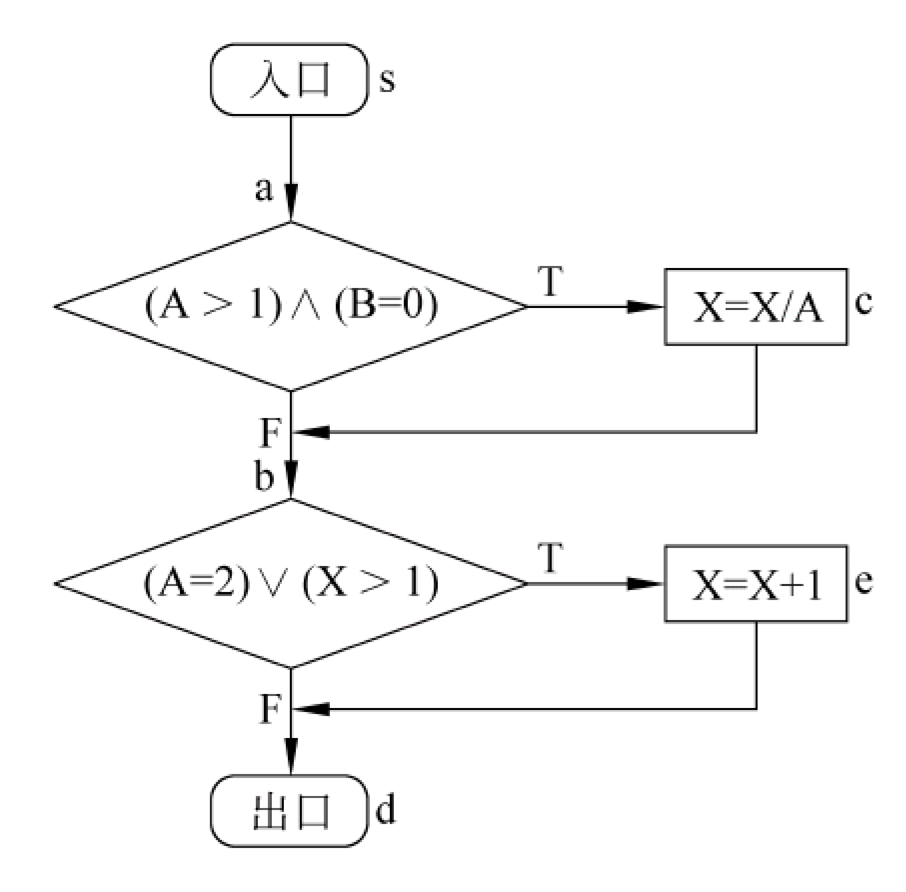


图 4-1 某被测程序流程图

语句覆盖是一种很弱的逻辑覆盖标准，它对程序的逻辑覆盖很少。对图 4-1 表示的源程序，语句覆盖只覆盖了两个判断均为真的情况，若某个判定的结果为假，则对应的操作有错也不可能通过语句覆盖发现；此外，语句覆盖只关心判定的结果，而没有考虑判定中的条件（注意本小结中的条件均指简单条件）及条件之间的逻辑关系。例如若在图 4-1 中将第一个判定中的符号 ∧ 错写为 ∨，或将第二个判定中的 X＞1 错写为 X＜1，则也不可能被 A＝2，B＝0，X＝4 这一组测试用例发现。

2. 判定覆盖

判定覆盖（Decision Coverage）的含义是，设计足够多的测试用例，使被测程序中的每个判定取到每种可能的结果，即覆盖每个判定的所有分支。故判定覆盖也称为分支覆盖。显然，若实现了判定覆盖，则必然实现了语句覆盖，故判定覆盖是一种强于语句覆盖的覆盖标准。

对图 4-1 表示的源程序，若要实现判定覆盖，则需覆盖 sacbed 和 sabd 两条路径，或覆盖 sacbd 和 sabed 两条路径，可设计如下两组测试用例：

A＝3，B＝0，X＝3（覆盖路径 sacbd）

A＝2，B＝1，X＝1（覆盖路径 sabed）

判定覆盖对程序的逻辑覆盖程度仍不高，图 4-1 表示的源程序有 4 条路径，但以上的测试用例只覆盖了其中的两条。

3. 条件覆盖

条件覆盖（Condition Coverage）的含义是，设计足够多的测试用例，使被测程序中的每个条件取到各种可能的结果。

对图 4-1 表示的源程序，考虑包含在两个判定中的 4 个条件，每个条件均可取真假两种值。若要实现条件覆盖，应使以下 8 种结果成立：

A＞1，A＝1，B＝0，B≠0，A＝2，A≠2，X＞1，X⩽1

这 8 种结果的前 4 种是在 a 点出现的，而后 4 种是在 b 点出现的。

为了覆盖这 8 种结果，可设计如下两组测试用例：

A＝2，B＝0，X＝4（覆盖 A＞1，B＝0，A＝2，X＞1，执行路径 sacbed）

A＝1，B＝1，X＝1（覆盖 A＝1，B≠0，A≠2，X＝1，执行路径 sabd）

条件覆盖一般比判定覆盖强，因为条件覆盖关心判定中每个条件的取值，而判定覆盖

只关心整个判定的取值。也就是说，若实现了条件覆盖，则也实现了判定覆盖，如上述两组测试用例也实现了判定覆盖。但这不是绝对的，某些情况下，也会有实现了条件覆盖却未能实现判定覆盖的情形。例如下面两组测试用例：

A=2,B=0,X=1(覆盖 A>1,B=0,A=2,X=1,执行路径 sacbed)

A=1,B=1,X=2(覆盖 A=1,B≠0,A≠2,X>1,执行路径 sabed)

此两组测试用例均使图 4-1 中第二个判定取值为真，而未覆盖到第二个判定取值为假的情况。

甚至可能出现这样的情况，对某被测程序实现了条件覆盖却未实现语句覆盖，读者可自行举例。

4. 判定—条件覆盖

既然实现了判定覆盖不一定能够实现条件覆盖，而实现了条件覆盖也不一定能够实现判定覆盖，故可设计更高的逻辑覆盖标准将两者兼顾起来，这就是判定—条件覆盖。

判定—条件覆盖的含义是，设计足够多的测试用例，使被测程序中的每个条件取到各种可能的结果，且每个判定取到各种可能的结果；若实现了判定—条件覆盖，则必然实现了判定覆盖和条件覆盖。

对图 4-1 表示的源程序，若要实现判定—条件覆盖，可设计如下两组测试用例：

A=2,B=0,X=4(覆盖 A>1,B=0,A=2,X>1,执行路径 sacbed)

A=1,B=1,X=1(覆盖 A=1,B≠0,A≠2,X=1,执行路径 sabd)

实际上，这两组测试用例就是先前我们为实现条件覆盖而设计的两组测试用例。

5. 条件组合覆盖

当某个判定中存在多个条件时，仅仅考虑单个条件的取值是不够的。条件组合覆盖的含义是，设计足够多的测试用例，使被测程序中每个判定的所有条件取值组合都至少出现一次。

对图 4-1 表示的源程序，若要实现条件组合覆盖，应使如下 8 种条件取值组合至少出现一次：

① A>1,B=0　　② A>1,B≠0

③ A=1,B=0　　④ A=1,B≠0

⑤ A=2,X>1　　⑥ A=2,X=1

⑦ A≠2,X>1　　⑧ A≠2,X=1

以上 8 种组合中，前 4 种组合是图 4-1 中第一个判定的条件取值组合，后 4 种组合则是第二个判定的条件取值组合。

为覆盖此 8 种组合，可设计如下的 4 组测试用例：

A=2,B=0,X=4(覆盖①、⑤两种组合，执行路径 sacbed)

A=2,B=1,X=1(覆盖②、⑥两种组合，执行路径 sabed)

A=1,B=0,X=2(覆盖③、⑦两种组合，执行路径 sabed)

A=1,B=1,X=1(覆盖④、⑧两种组合，执行路径 sabd)

对某被测程序，若实现了条件组合覆盖，则一定实现了判定覆盖、条件覆盖及判定—

条件覆盖。但条件组合覆盖不一定能覆盖程序中的每条路径，如上述 4 组测试用例就没有覆盖到图 4-1 所示源程序中的路径 sacbd。

6. 修正条件判定覆盖

修正条件判定覆盖是由欧美的航空/航天制造厂商和使用单位联合制定的“航空运输和装备系统软件认证标准”，目前在国外的国防、航空、航天领域应用广泛。这个覆盖度量需要足够的测试用例来确定各个条件是否能够影响到包含的判定的结果。它要求满足两个条件：首先，每一个程序模块的入口和出口点都要至少被调用一次，每个程序的判定到所有可能的结果值要至少转换一次；其次，程序的判定被分解为通过逻辑操作符（AND，OR）连接的布尔条件，每个条件对于判定的结果值是独立的。

实现修正条件判定覆盖，需要付出极大的成本（通常能够支持修正条件判定覆盖的测试工具价格极其昂贵）。

读者可思考，为实现修正条件判定覆盖，如何对图 4-1 表示的源程序设计测试用例。

4.2.5 基本路径测试法

1. 基本路径测试的概念

图 4-1 所示源程序共有 4 条路径，若要覆盖此 4 条路径，可设计如下 4 组测试用例：

A=2，B=0，X=4（覆盖路径 sacbed）

A=2，B=1，X=1（覆盖路径 sabed）

A=1，B=1，X=1（覆盖路径 sabd）

A=3，B=0，X=1（覆盖路径 sacbd）

但必须看到，图 4-1 所示源程序是一个极简单的程序，大多数情况下，由于程序中选择结构和循环结构的存在，使得测试程序中的每一条路径是现实不允许的事情。故必须将测试的路径数目压缩到一定范围内，基本路径测试（Basic Path Testing）就是这样的一种测试。它在程序的流图基础上，确定程序的环路复杂性，导出基本路径的集合，进而在其基础上设计测试用例，这些测试用例能覆盖到程序中的每条可执行语句。基本路径测试法也可认为是逻辑覆盖法的一种覆盖标准。

2. 流图

流图也称为程序图，它将流程图中的结构化构件改用一般有向图的表示形式，如图 4-2 所示。

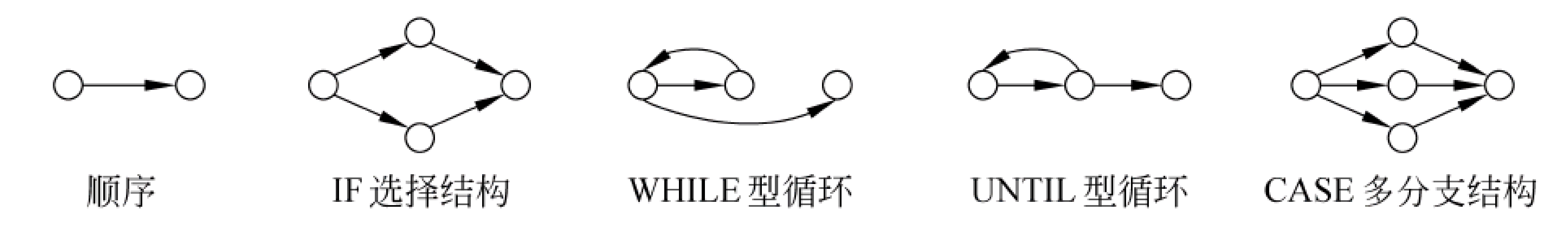
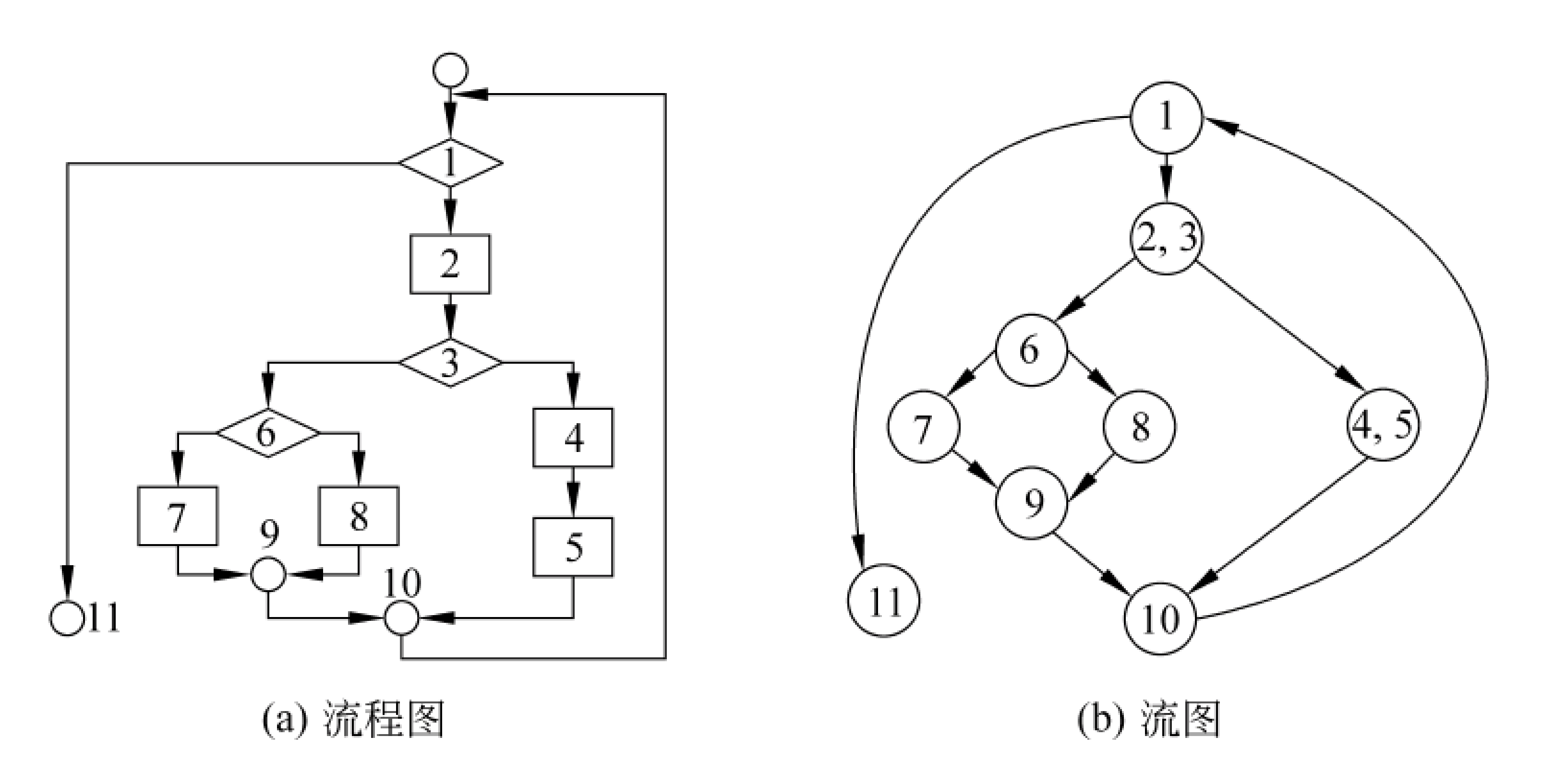


图 4-2 结构化构件在流图中的表示形式

在图 4-2 中，每个圆圈代表一个结点，表示源程序或 PDL（Procedure Design Language，过程设计语言）中的一个或多个连续的无分支语句。

例如，图 4-3(a)所示的流程图（假设每个判断均不含复合条件），它所对应的流图如

图 4-3(b)所示。

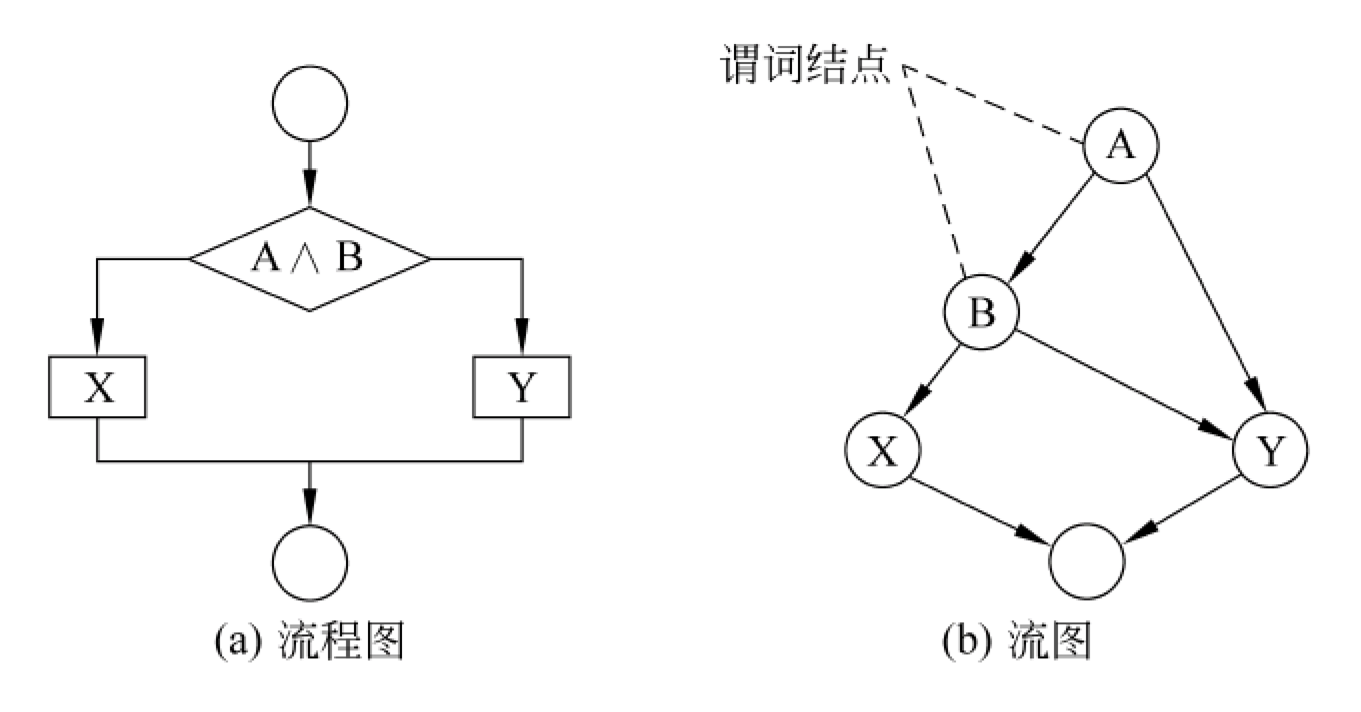


(a) 流程图　(b) 流图

图 4-3　流程图和对应的流图

流图中用谓词结点来表示简单条件判断，即谓词结点不允许含有复合条件。对于程序(或流程图)中的复合条件，应将其转化为多个简单条件判断，在流图中用相应的谓词结点表示。

如图 4-4(a)所示的流程图，图中的判断含有两个条件，即为复合条件判断，故将此判断在流图中用两个谓词结点表示。图 4-4(a)流程图对应的流图如图 4-4(b)所示。



(a) 流程图　(b) 流图

图 4-4　流程图及其对应的流图

3. 程序的环路复杂性

程序的环路复杂性(Cyclomatic Complexity)又称为圈复杂性，其值等于流图中的区域个数。在进行基本路径测试时，确定了程序的环路复杂性，则可在其基础上确定程序基本路径集合的独立路径数目，这个数目是确保程序中每条可执行语句至少执行一次的测试用例数目的最小值。

独立路径是一条含有以前未处理的语句或判断的路径。在流图中，独立路径表现为至少含有一条其他独立路径中均没有的边的路径。

图 4-3(b)所示的流图包含 4 个区域，故所对应的程序的环路复杂性度量 V(G)为 4，程序有以下 4 条独立路径。

路径 1：1—11。

路径 2：1—2—3—4—5—10—1—11。

路径 3：1—2—3—6—8—9—10—1—11。

路径 4：1—2—3—6—7—9—10—1—11。

可由此设计测试用例，覆盖以上 4 条独立路径，即可使程序中的所有可执行语句和所有判断的真、假分支至少执行一次。

利用以下公式也可以计算程序的环路复杂性度量 V(G)：

$$V(G)=E-N+2$$

式中 E 为流图中的边数，N 为流图中的结点数。如图 4-3(b)所示的流图中，边数为 11，结点数为 9，故 V(G)＝11－9＋2＝4。

此外，还可利用流图中的谓词结点数来计算环路复杂性度量 V(G)：

$$V(G)=P+1$$

式中 P 为流图中的谓词结点数。如图 4-3(b)所示的流图中，谓词结点数为 3，故 V(G)＝3＋1＝4。若能事先确定程序中的简单条件判断的个数（即等于流图中的谓词结点数），则可在不画出流图的情况下确定程序的环路复杂性。

4. 基本路径测试范例

下面给出了一个利用基本路径测试法设计测试用例的例子。

试以如下 PDL 为基础，利用基本路径测试法设计测试用例，该 PDL 代表的程序允许最多输入 100 个数（以－999 作为输入结束标志），统计有效输入值（即数值在规定的最小值和最大值之间）的平均值，若有效输入值的个数为 0，则平均值计为－999。

```
PROCEDURE average
  INTERFACE RETURNS average;
  INTERFACE ACCEPTS value,min,max;
  TYPE value[1:100] IS SCALAR ARRAY;
  TYPE average, total.input,total.valid, min,max,sum IS SCALAR;
  TYPE i IS INTEGER;
  i=1;
  total.input=total.valid=0;
  sum=0;
  DO WHILE value[i]<>-999 AND total.input<100
    total.input 加 1;
    IF value[i] >=min AND value[i] <=max
      THEN
        total.valid 加 1;
        sum=sum+ value[i];
      ELSE skip;
    ENDIF
    i 加 1;
  ENDDO
  IF total.valid>0
    THEN average=sum/ total.valid;
    ELSE average=-999;
  ENDIF
END average
```

此题的完成步骤如下：

（1）根据 PDL 画出程序的流图。为画出流图，先为 PDL 中的语句和判断编号，用以确定流图中的结点，对语句和判断编号后的 PDL 如图 4-5 所示，根据图 4-5 画出的流图如图 4-6 所示。

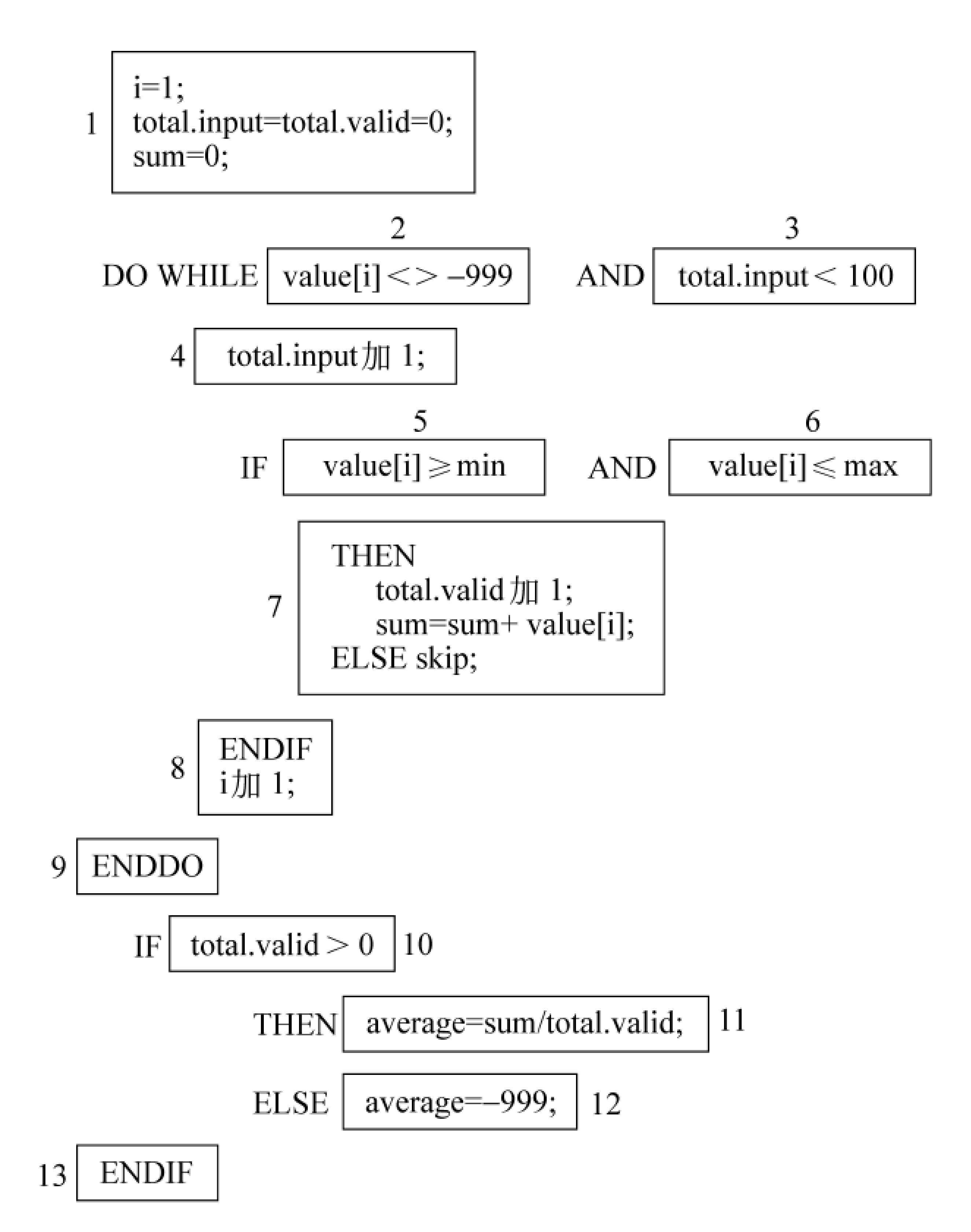


图 4-5　对语句和判断编号后的 PDL

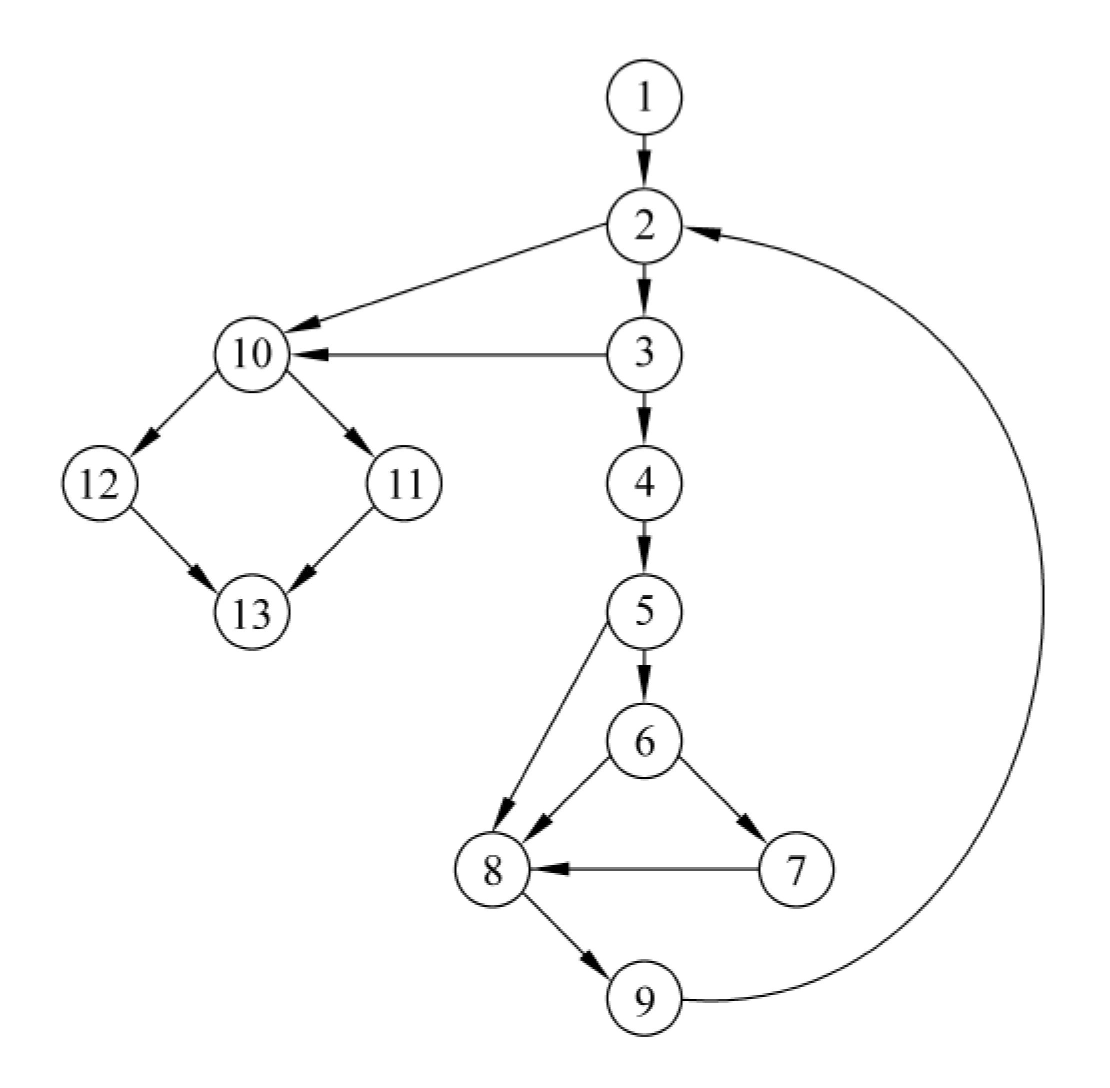


图 4-6　过程 average 的流图

（2）确定程序的环路复杂性。可按以下3种方法确定环路复杂性。

① 按流图的区域个数计算：V(G)＝6。

② 按公式V(G)＝E－N＋2计算：V(G)＝17－13＋2＝6。

③ 按公式V(G)＝P＋1计算：V(G)＝5＋1＝6。

（3）确定程序的基本路径集合。由于基本路径集合中的独立路径数目等于程序的环路复杂性，故为6。可确定如下6条独立路径。

路径1：1—2—10—11—13

路径2：1—2—10—12—13

路径3：1—2—3—10—11—13

路径4：1—2—3—4—5—8—9—2…

路径5：1—2—3—4—5—6—8—9—2…

路径6：1—2—3—4—5—6—7—8—9—2…

路径4,5,6后的省略号表示路径中的后续结点对于测试用例的设计并不重要。值得注意的是，对于同一个程序，基本路径集合不是唯一的。

明确谓词结点将有助于测试用例的设计，图4-6中的谓词结点为2,3,5,6,10。

（4）设计测试用例，使基本路径集合中的每条独立路径至少执行一次。

可以设计如下测试用例。

测试用例1（覆盖路径1的测试用例）

value[k]≠－999且至少有一个value[k]是有效输入值，k<i；
value[i]＝－999，2≤i≤100；

预期结果：正确计算出有效输入值的平均值。

测试用例2（覆盖路径2的测试用例）

value[1]＝－999；

预期结果：计算出的平均值为－999。

测试用例3（覆盖路径3的测试用例）

试图输入101个数值，前100个数值均不为－999且至少有一个为有效输入值，第101个数值也不为－999。

预期结果：正确计算出有效输入值的平均值。

测试用例4（覆盖路径4的测试用例）

value[k]≠－999且至少有一个value[k]小于min，k<i；
value[i]＝－999，2≤i≤100；

预期结果：正确计算出平均值。若有效输入值的个数为0，则平均值为－999。

测试用例5（覆盖路径5的测试用例）

value[k]≠－999且至少有一个value[k]大于max，k<i；
value[i]＝－999，2≤i≤100；

预期结果：正确计算出平均值。若有效输入值的个数为0，则平均值为－999。

测试用例 6（覆盖路径 6 的测试用例）

value[k]≠－999 且至少有一个 value[k]是有效输入值，k＜i；
value[i]＝－999，2≤i≤100；

预期结果：正确计算出有效输入值的平均值。

4.2.6　控制结构测试

基本路径测试可以认为是控制结构测试中的一种，下面介绍控制结构测试中的一些其他测试策略。

1. 分支结构的路径数的确定

当程序中的判定多于一个时，分支结构可分为两类：嵌套型分支结构和连锁型分支结构，如图 4-7 所示。

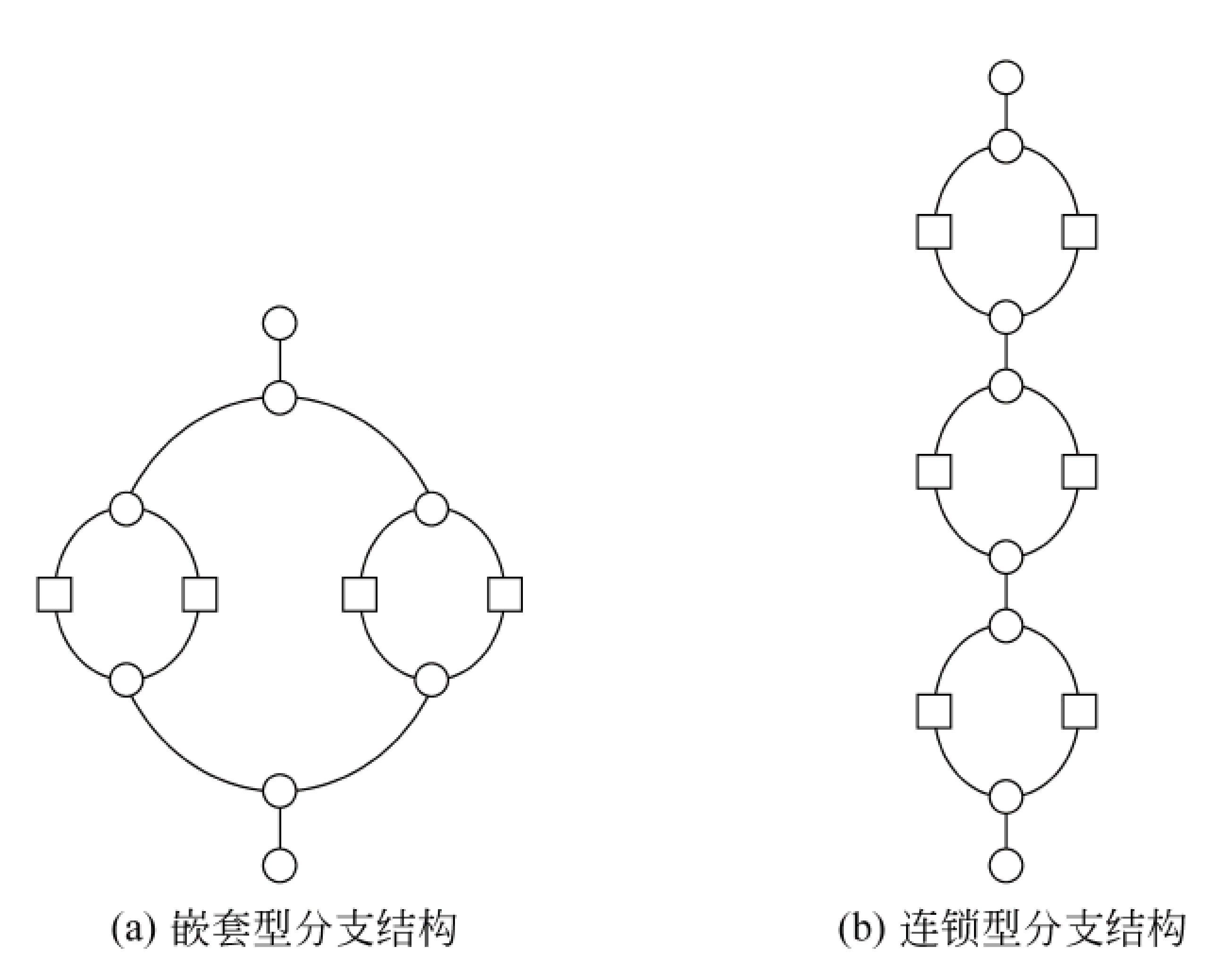


图 4-7　分支结构的两种类型

对于嵌套型分支结构，若有 n 个判定语句，需要 $n+1$ 个测试用例去覆盖其 $n+1$ 条路径；对于连锁型分支结构，若有 n 个判定语句，则需要 2^n 个测试用例去覆盖其 2^n 条路径。当 n 较大时，测试的工作量将变得难以承受。

为减少测试用例的数目，可采用构造正交表方法选取部分路径进行测试。在假定各条路径的重要性相同，或不清楚各路径重要性的情况下，可进行均匀抽样；若能确定各路径的重要性，则可通过对路径进行加权，从而筛选掉部分路径，再使用均匀抽样方法。抽样的具体步骤如下：

① 设连锁型分支结构中含有 n 个判定，计算满足关系式 $n+1\leqslant 2^m$ 的最小自然数 m。

② 设 $t=2^m$，构造正交表 L_t。

③ 利用正交表 L_t 选取测试路径。

下面我们来看一个例子。

一个连锁型分支结构中有 3 个判定语句 P_1，P_2，P_3，故共有 2^3 即 8 条路径，为减少测试需覆盖的路径数目，可采用构造正交表的方法选取部分路径进行测试，步骤如下。

① 计算满足关系式 $n+1\leqslant 2^m$ 的最小自然数 m。此例中，$n=3$，故 $m=2$。

② 计算 t，$t=2^m=4$，构造正交表 L_4（正交表的构造过程从略，可参看相关书籍）。正交表 L_4 如表 4-1 所示。

表 4-1　正交表 L_4

行号 \ 列号	1	2	3
1	0	0	0
2	1	0	1
3	0	1	1
4	1	1	0

正交表 L_4 中的每一列表示一个判定，第 1，2，3 列分别对应于判定 P_1，P_2，P_3，每一行为一个可取的测试用例。

③ 判定 P_1 的真、假分支分别记为 S_1，S_2；判定 P_2 的真、假分支分别记为 S_3，S_4；判定 P_3 的真、假分支分别记为 S_5，S_6。用判定的取真分支代替正交表 L_4 中的"1"，用判定的取假分支代替正交表 L_4 中的"0"，则确定了 4 条测试路径，正交表 L_4 的第 1，2，3，4 行代表的测试用例对应的路径分别为 $S_1—S_3—S_5$，$S_2—S_3—S_6$，$S_1—S_4—S_6$，$S_2—S_4—S_5$。

通过该方法的运用，将测试用例的数目（即为测试路径的数目）从 8 减为 4，简化了测试用例的设计。

2. 对循环结构的测试

绝大多数程序都要用到循环结构，对循环结构进行测试的目的是发现循环结构中的错误。循环可分为简单循环、嵌套循环、并列循环和非结构循环几种类型，如图 4-8 所示。

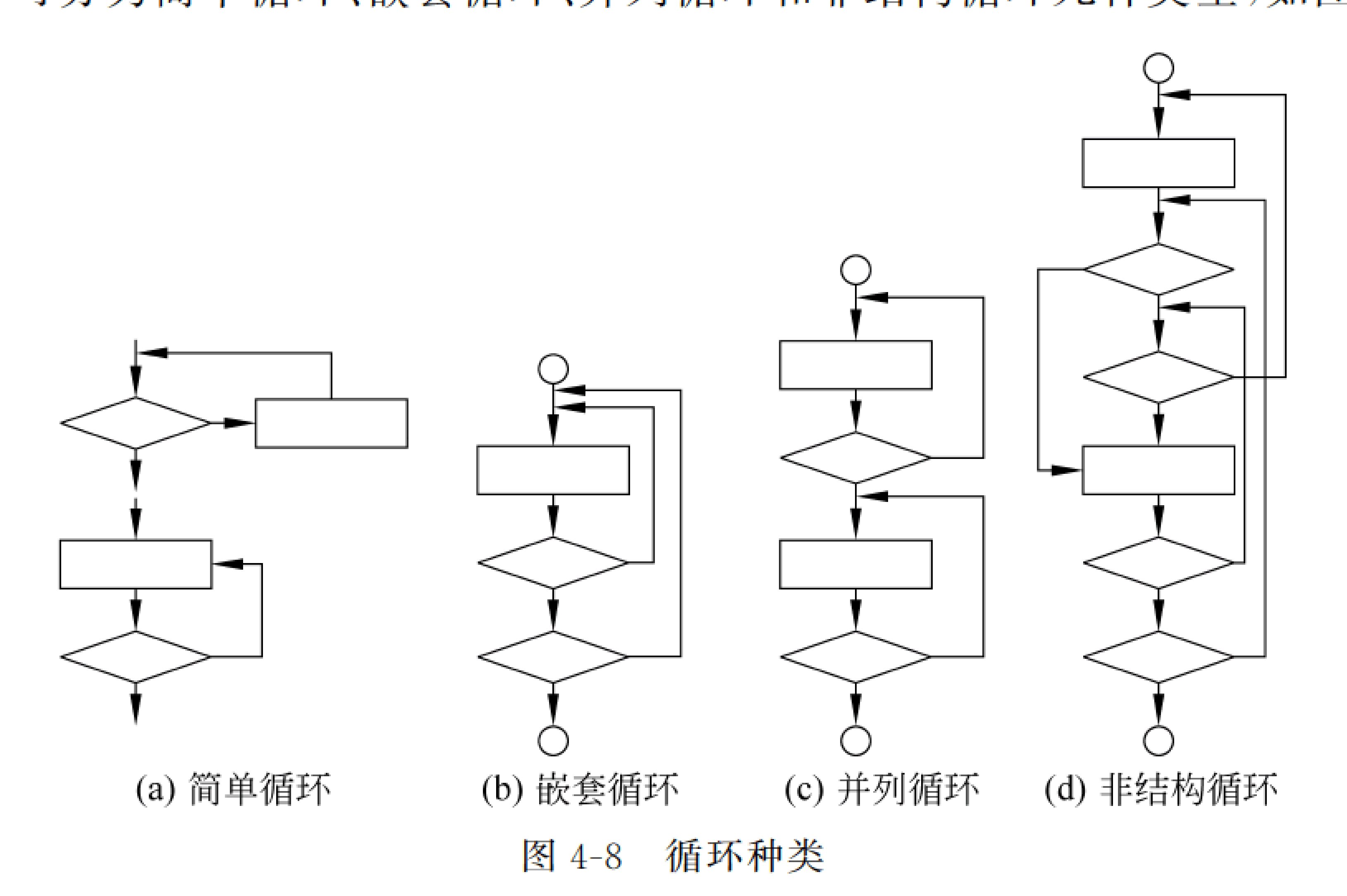


图 4-8　循环种类

下面分别讨论对每一类循环的测试策略。

（1）对简单循环的测试

对于循环次数不大于 n 的简单循环，可进行如下测试。

① 不执行循环。
② 执行一次循环。
③ 执行两次循环。
④ 执行 m 次循环，$m<n$。
⑤ 执行 $n-1$ 次循环。
⑥ 执行 n 次循环。
⑦ 执行 $n+1$ 次循环。

(2) 对嵌套循环的测试

若按照简单循环的测试策略对嵌套循环进行测试，会使测试次数呈几何级数增长。为减少对嵌套循环的测试次数，可使用如下策略：

① 从最内层循环开始测试，此时所有外层循环的次数都取最小值，内层循环按简单循环的测试策略进行测试。

② 从内向外，退到外一层循环进行测试，该层循环的所有外层循环次数仍取最小值，而该层循环的内层循环次数则可取某个典型值。

③ 继续向外扩展，对每一个循环层次按②中所述方法进行测试，直至所有循环层次均被测试完。

(3) 对并列循环的测试

若并列的两个循环结构完全独立，可使用简单循环的测试策略；若两个循环结构不独立，如第一个循环的计数值作为第二个循环的初值，则应使用嵌套循环的测试策略。

(4) 对非结构循环的测试

对于非结构循环，应先将其转化为结构化循环，再使用上述的测试策略进行测试。

还有一些测试策略也属于控制结构测试的范畴，如对复合条件的测试、数据流测试等，在此从略，有兴趣的读者可进一步参看其他资料。

4.2.7　程序插桩

程序插桩(Instrumentation)是一种通过向被测程序中插入操作来发现和定位错误的方法，在程序测试和调试(调试即在发现程序存在运行错误以后，寻找错误的原因和位置并排除错误)中有着广泛的应用。

在测试或调试程序时，常常需要在程序中插入一些打印语句，使其在程序执行过程中能打印出我们关心的信息，通过这些信息来了解程序执行过程中的动态特性，如程序的实际执行路径，程序中某语句的执行次数，程序中各路径的被覆盖程度，程序中某变量在特定时刻的值等，这些都是可以通过程序插桩来完成的。

下面以计算整数 X 和整数 Y 的最大公约数的程序为例，说明程序插桩方法的要点。计算整数 X 和整数 Y 的最大公约数的程序流程图如图 4-9 所示。

图 4-9 中的虚线框是为了记录语句的执行次数而插入的，其形式为

$$C(i) = C(i) + 1 \quad i = 1,2,\cdots,6$$

程序从入口开始执行到出口结束执行，经过的计数语句记录下该程序点语句的执行次数。若在程序的入口处插入了对计数器 $C(i)$ 初始化的语句，在程序出口处则插入打印

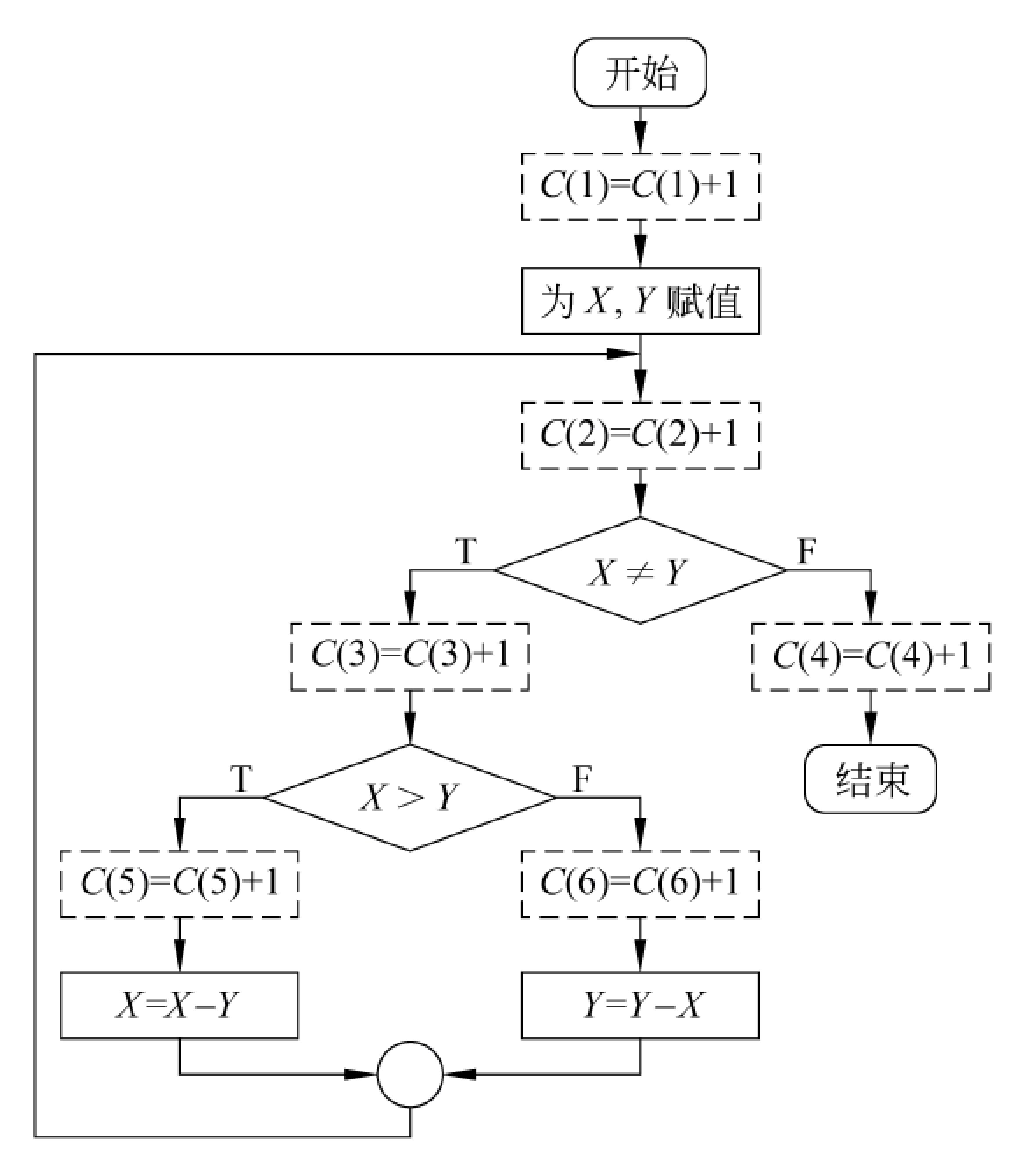


图 4-9 插桩后的求最大公因数程序流程图

这些计数器的语句,构成了一个完整的插桩程序。

显然,通过使用程序插桩方法,测试人员不仅能像以前一样得到程序执行的结果数据,还能跟踪程序的执行流程,深入了解程序执行过程中的动态特性,这使得测试或调试更富有成效。

设计程序插桩方法时需要考虑如下问题:

- 应探测程序中的哪些信息;
- 在程序的什么位置设置探测点。

这两个问题均需要结合具体的程序进行考虑,但对于第二个问题,有一个准则对解决问题有一定的帮助,那就是在一般情况下,在没有分支结构的多条连续语句构成的一段程序中,只需插入一个计数语句。这样有利于将探测点的数目减少到最小。当然,若程序中出现多种控制结构,整个结构较为复杂时,应针对具体的程序结构进行分析,以确定在何处插入计数语句。

4.3 白盒测试方法的综合使用策略

在白盒测试中,应灵活地使用各种测试方法。使用白盒测试方法对软件进行测试的综合策略如下。

(1) 静态测试和动态测试的时序关系

一般可先进行静态测试,即使用代码检查法、静态结构分析法、代码质量度量法等进行测试;接着进行动态测试,即使用逻辑覆盖法、基本路径测试法、控制结构测试、程序插桩等方法进行测试。当然,这不是绝对的。

(2) 白盒测试的重点

覆盖率测试是白盒测试的重点,一般可使用基本路径测试方法使基本路径集合中的每条独立路径至少执行一次。对于重要的程序模块,应使用多种覆盖率标准衡量对代码的覆盖率。对于各种覆盖率标准,读者可进一步参考其他相关资料。

(3) 不同测试阶段使用的白盒测试方法

在不同的测试阶段,使用的白盒测试方法不尽相同。

① 在单元测试阶段,以代码检查法、逻辑覆盖法和基本路径测试法为主。

② 在集成测试阶段,需增加静态结构分析法和代码质量度量法。

③ 在集成测试之后的测试阶段,应尽量使用黑盒测试方法,但若发现了软件中的严重问题且无法用黑盒测试方法定位,则仍需选择性地使用白盒测试方法,深入到模块的内部以定位错误。

4.4　对黑盒、白盒测试方法的总结

通过第 3 章和第 4 章的讲述,对黑盒测试和白盒测试应该有了一个较为清楚的认识。黑盒测试和白盒测试作为两种出发点完全不同的测试方法,各有其特点,在软件测试中是缺一不可的。

黑盒测试完全不考虑程序的具体实现过程,从程序外部对其功能、性能等进行测试,故可以认为是站在用户角度进行的测试。由于黑盒测试具有成本较低的优点,被测试从业人员广泛采用。

但黑盒测试有其不足之处,如对特定的输入,软件的输出恰巧是正确的,但内部的运算有错,黑盒测试就无法发现;还有如软件中存在的内存泄漏、误差累积等隐患也是黑盒测试无能为力的。

与黑盒测试相比,白盒测试深入到程序的内部进行测试,能发现比黑盒测试更多的错误,也更易于定位错误的原因和具体位置,并能得出测试对代码的覆盖率,弥补了黑盒测试只能从程序外部进行测试且难于衡量测试完整性的不足。

虽然白盒测试的优点很多,但不能对一个测试项目盲目地、无限制地施用白盒测试方法,因为白盒测试与黑盒测试的方式虽然不同,但在很多场合,会与黑盒测试产生同样的效果,应减少此类冗余的测试,毕竟,白盒测试意味着更多的测试成本。

一般说来,在软件测试的单元测试阶段,以使用白盒测试法为主对被测单元进行测试;在集成测试阶段,可使用黑盒、白盒相结合的方法测试多个单元组装在一起能否按预期的设计要求工作,这种测试策略也可以理解为灰盒测试方法;在集成测试之后的测试阶段,目标软件已基本成型,应使用黑盒测试方法对软件进行测试。

4.5　小结

本章介绍了白盒测试方法的含义,及各种典型的白盒测试方法。白盒测试的方法很多,应根据被测项目的具体情况灵活选择。白盒测试作为一种基于程序内部结构进行的

测试，涉及的理论很多，在这一章，一些涉及较深理论的方法或准则被略去了，如域测试、符号测试、Z路径覆盖、程序变异等白盒测试方法，又如白盒测试中的一些测试覆盖准则，以及静态结构分析的具体实施方法等，但这不代表它们对白盒测试是不重要的，读者可进一步参阅相关资料。

习　题

1. 白盒测试的含义是什么？有何优缺点？

2. 为何白盒测试无法实现穷举测试？

3. 白盒测试方法主要包括哪几类？

4. 对某被测程序，若实现了条件组合覆盖，则一定实现了判定覆盖、条件覆盖及判定—条件覆盖，为什么？

5. 阅读下列代码（由C语言书写）后回答问题。

```
int IsLeap(int year)
{
  if (year % 4 == 0)
  {
    if (year % 100 == 0)
    {
      if ( year % 400 == 0)
        leap = 1;
      else
        leap = 0;
      }
    else
        leap = 1;
  }
  else
      leap = 0;
  return leap;
}
```

（1）画出上述代码的流图。

（2）计算上述代码的环路复杂性V(G)。

（3）假设输入的取值范围是1000＜year＜2001，试用基本路径测试法为变量year设计测试用例，使其满足基本路径覆盖的要求。

6. 程序插桩的作用是什么？

7. 白盒测试方法的综合使用策略是什么？

8. 在软件测试中，应如何将黑盒、白盒测试方法结合起来使用，试谈谈你的看法。

第 5 章

单 元 测 试

本章要点：

- 单元测试的含义、目的和优点。
- 单元测试使用的方法。
- 单元测试的步骤。
- 单元测试环境。
- 驱动模块和桩模块的含义。
- 单元测试用例的设计。
- CppUnit 的安装及环境配置。
- CppUnit 的使用步骤。

单元测试是软件测试中最基本的一种测试类型，它能以较高的效率和较小的成本发现软件中的错误。在这一章，将介绍单元测试的基本概念，包括其含义、目的、方法、步骤和环境等，还将详细介绍单元测试的内容及单元测试用例的设计思路，以及如何用 CppUnit 进行单元测试。

5.1 单元测试概述

1. 单元测试的概念

什么是单元测试(Unit Testing)？单元测试是对软件基本组成单元的测试。单元的具体含义是什么呢？一般认为，在传统的结构化编程语言中，比如 C 语言，要进行测试的单元一般是模块，也就是函数或子过程；在像 C ++ 这样的面向对象的语言中，要进行测试的基本单元是类或类的方法；对 Ada 语言来说，开发人员可以选择是在独立的过程和函数，还是在 Ada 包的级别上进行单元测试；单元测试的原则同样被扩展到第四代语言(4GL)的开发中，在这里基本单元往往对应为一个菜单或显示界面。

当然，在单元测试过程中应灵活把握单元的概念。比如，在对 C 语言代码的单元测试中，若某函数 A 仅被函数 B 调用，且函数 A 和函数 B 的代码在一定的范围内，则可将函数 A 和函数 B 作为一个被测的单元，但必须在单元测试方案中明确地加以说明。

由于单元测试是对单个被测模块的测试，故多个被测模块之间的单元测试可以同时进行，以提高单元测试的效率。

与其他测试不同，单元测试可看作是编码工作的一部分，一般应该由编程人员完成，有时测试人员也加入进来，但编程人员仍会起到主要作用。也就是说，编程人员有确保自己编写的软件单元准确的责任。

单元测试是软件开发过程中级别最低的一种测试，但也是十分重要的一种测试。单元测试的依据是软件的详细设计描述、源程序清单、编码标准等。

2. 单元测试的目的

确保被测单元的代码正确是单元测试的主要目标。具体说来，单元测试的目的主要包括如下几方面：

(1) 验证代码能否达到详细设计的预期要求。

(2) 发现代码中不符合编码规范的地方。

(3) 准确定位发现的错误，以便排除错误。

3. 单元测试的优点

一旦对软件中各单元的编码工作完成，开发人员总是迫切地希望进行软件单元间的集成，这样就能够看到实际系统的运行效果。但由于要进行单元测试，将推迟对整个软件系统进行联调的启动时间。因而，目前有一部分人对单元测试抱有偏见，认为单元测试浪费了太多的时间，推迟了系统的交付。

实际上，若不进行单元测试，或对单元测试敷衍了事，各单元集成后能够正常工作的可能性是很小的，将有可能出现各种各样的问题。事实上，这些问题往往不是大的问题，大都可以在严格进行的单元测试阶段发现并解决。但在集成阶段或系统测试阶段，若要定位并解决软件中的这些问题，会比在单元测试阶段解决付出更大的成本，还很可能造成软件交付时间的推迟。

总的说来，单元测试的优点主要表现为以下两方面：

(1) 由于单元测试是在编码过程中进行的，若发现了一个错误，不管是从做回归测试的角度，还是对错误原因理解的深刻性的角度出发，修复错误的成本远小于集成测试阶段，更小于系统测试阶段。

(2) 在编码的过程中考虑单元测试问题，有助于编程人员养成良好的编程习惯，提高源代码的质量。

基于单元测试具有的优点，开发人员应当高度重视单元测试，有计划地、严格地对被测单元实施单元测试，以提高各单元的质量，提高软件开发的效率。

4. 单元测试的测试方法

在单元测试阶段，应使用白盒测试方法和黑盒测试方法对被测单元进行测试，其中以使用白盒测试方法为主。

单元测试阶段应使用黑盒测试方法，从外部接口处对被测单元进行测试，以验证其能否完成规格说明中的预期功能；在单元测试阶段还应使用代码检查法、逻辑覆盖法、基本路径测试法等白盒测试方法，深入被测单元的内部，对被测单元进行静态和动态测试，以

验证代码是否符合详细设计的要求及一定的编码规范。

在单元测试阶段以使用白盒测试方法为主，是因为在单元测试阶段白盒测试消耗的时间、人力、物力等成本一般会大于黑盒测试的成本，而并不是说黑盒测试在单元测试中是不重要的。

5.2　单元测试的步骤

单元测试的实施应遵循一定的步骤，力争做到有计划、可重用。

所谓单元测试的可重用，是指单元测试不仅仅是作为无错编码的一种辅助手段在一次性开发过程中使用，当软件修改或移植到新的运行环境时，单元测试对应的测试用例及测试脚本(即自动测试的程序代码)应可被重复使用。因此，所有的单元测试都必须在软件系统的整个生命周期中进行维护。

单元测试的步骤如下：

① 计划单元测试。确定测试内容，初步制定测试策略，确定测试所用的资源，安排测试的进度。

② 设计单元测试。创建单元测试环境，制定测试方案，细化测试过程。

③ 实现单元测试。编写测试用例及测试脚本。

④ 执行单元测试。对被测单元执行测试用例及测试脚本，记录被测单元的执行过程和发现的错误，定位和排除错误。

⑤ 单元测试结果分析并提交测试报告。对单元测试的结果进行分析、归类，确认单元测试是否完备，并编制和提交单元测试报告。

5.3　单元测试环境

单元测试中的被测单元往往不是一个可以独立运行的程序，故在执行单元测试阶段的动态测试时，应建立单元测试的环境，使被测模块能够运行起来，以达到对其进行测试的目的。在建立单元测试环境的过程中，人们很可能面临开发驱动模块(Driver)和桩模块(Stub)的任务。

何谓驱动模块和桩模块？驱动模块是用来代替被测单元的上层模块的。驱动模块能接收测试数据，调用被测单元，也就是将数据传递给被测单元，最后打印测试的执行结果。驱动模块可以理解为被测单元的主程序。

下面来看一个驱动模块的例子。

被测程序如下。

```
int Fun(int in)
{
    if (in >= 0)
    {
        return 1;
    }
```

```
    else
    {
        return -1;
    }
}
```

用 TCL 脚本语言进行扩展指令编写时,针对该被测函数,驱动程序如下。

```
int Ex_TestFun(ClientData clientData,Tcl_Interp * interp, int argc,char * argv[])
{
    int i;
    int ret, iExceptedRet;

    //打开测试结果记录文件
    FILE * out;
    out = fopen("D:\\result.txt","a");

    //第 1 步:检查用户输入参数个数是否正确
    if (3 != argc)
    {
        fputs("Parameters error",out);
        fflush(out);
        return TCL_ERROR;
    }

    //第 2 步:取出用户输入参数
    if (TCL_OK != Tcl_GetInt(interp,argv[1],&i))
    {
        return TCL_ERROR;
    }

    if (TCL_OK != Tcl_GetInt(interp,argv[2],&iExceptedRet))
    {
        return TCL_ERROR;
    }

    //第 3 步:将参数传递给被测函数
    ret = Fun(i);

    //第 4 步:将被测函数执行结果和输入的期望结果进行比较,根据比较结果作为用例执行结
      果输出到测试报告中
      if (ret != iExceptedRet)
      {
        fputs("test fail",out);
        fflush(out);
      }
      else
      {
        fputs("test success",out);
        fflush(out);
```

```
        }

        return TCL_OK;
    }
```

至于桩模块，又称为存根模块，它用来代替被测单元的子模块。设计桩模块的目的是模拟实现被测单元的接口。桩模块不需要包括子模块的全部功能，但应做少量的数据操作，并打印接口处的信息。

下面来看一个桩模块的例子。

若被测程序如下。

```
int Max()
{
    int a = fun1();
    int b = fun2();
    if(a>b)
        return 1;
    else
        return 0;
}
```

则需要为语句 int a = fun1()和 int b = fun2()打桩，也就是设计桩模块，模拟函数 fun1()和 fun2()的返回值。

由于驱动模块和桩模块不是最终可提交的模块，因此人们在进行单元测试时应尽量避免开发驱动模块和桩模块。尤其应避免开发桩模块，因为驱动模块开发的工作量一般少于桩模块。其实道理是显而易见的，驱动模块是用来代替被测模块的实际上层模块的，只用于完成调用被测模块的工作，而桩模块要做的事情则要多些。

若采用自底向上的方式进行开发，底层的单元先开发并先测试，可以避免开发桩模块。采用这种方法测试上层单元时，也是对下层单元的间接测试，但当下层单元被改动后，则需要执行测试判断其上层单元是否需要修改。

当不得不开发驱动模块及桩模块时，则应力求简单以提高工作效率。但遗憾的是，过于简单的驱动模块和桩模块会影响单元测试的有效性，因而对被测单元的彻底的测试有时会被推迟到集成测试阶段完成。

建立单元测试的环境，需完成以下工作：

- 构造最小运行调度系统，即构造被测单元的驱动模块；
- 模拟被测单元的接口，即构造被测单元调用的桩模块；
- 模拟生成测试数据及状态，为被测单元运行准备动态环境。

单元测试环境可用图 5-1 简要地表示。

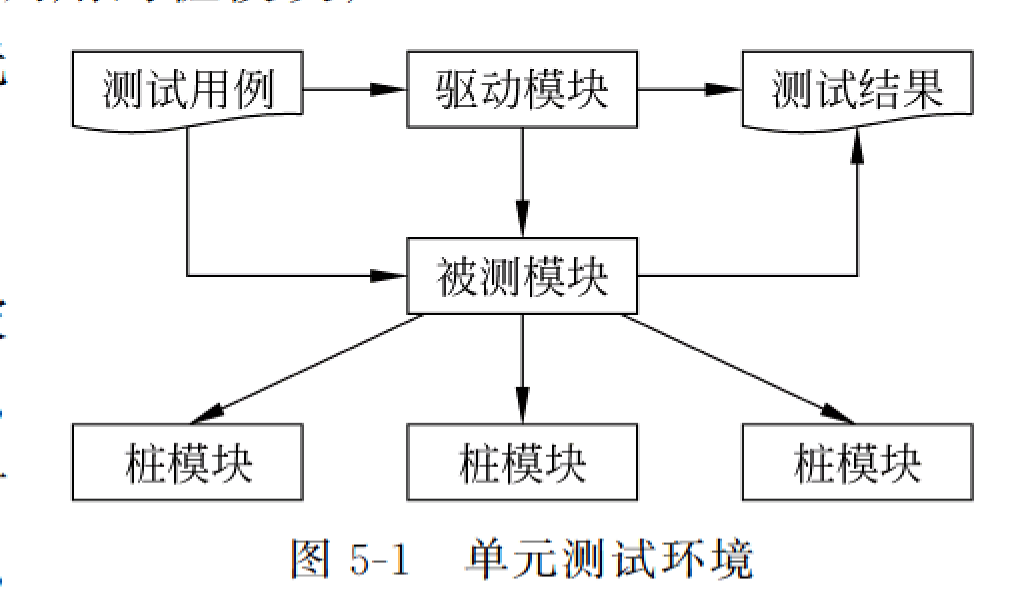


图 5-1　单元测试环境

单元测试环境时，不仅能成功地实施对被测单元的测试，还应能提供对测试过程的支持，如对测试执行过程的跟踪、测试结果的保留、对测试覆盖率的记录等。若要得出测试覆盖率，

可事先提出测试应达到的覆盖标准以指导测试用例的设计，并采用程序插桩方法，通过工具对被测单元进行插桩，在执行测试后统计测试覆盖率。

单元测试结论的有效性与单元测试环境模拟目标环境的精确性直接相关，故构造单元测试环境时，应尽可能考虑各种环境因素，如所有的隐性输入（系统时钟、文件状态、单元加载地点等）应被考虑，此外实际环境的代表物（相同的编译器、加载者、操作系统、计算机、输入分布等）也应被考虑。

5.4 单元测试用例的设计

5.4.1 单元测试的内容

单元测试应主要从单元接口、局部数据结构、独立路径、出错处理、边界条件几个方面对被测单元进行检查，如图 5-2 所示。

1. 单元接口

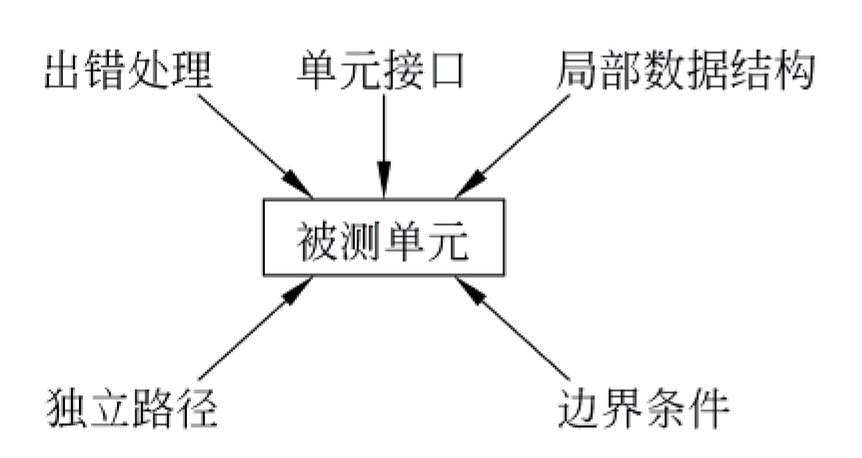


图 5-2 单元测试的内容

对单元接口的测试应主要考虑如下方面：

- 传递给被测单元的参数与被测单元的形式参数在属性、个数、顺序上是否一致。
- 被测单元调用子模块时，传递给子模块的参数与子模块的形式参数在属性、个数、顺序上是否一致。
- 是否修改了只做输入用的形式参数。
- 输出给标准函数的参数与标准函数的形式参数在属性、个数、顺序上是否一致。
- 全局变量的定义在各模块中是否一致。
- 约束条件是否通过形式参数进行传递。

若模块通过外设进行 I/O 操作时，还应该考虑下列因素：

- 文件属性是否正确。
- OPEN 与 CLOSE 语句是否正确。
- 规定的 I/O 格式说明与 I/O 语句是否匹配。
- 缓冲区容量与记录长度是否匹配。
- 在进行读/写操作之前是否打开了文件。
- 在结束文件处理时是否关闭了文件。
- 正文书写/输入错误是否存在。
- I/O 错误是否检查并做了处理。

2. 局部数据结构

对被测单元的局部数据结构测试应主要考虑如下方面：

- 数据类型说明是否正确。
- 是否使用了尚未赋值或尚未初始化的变量。
- 是否使用了错误的初始值或错误的默认值。

- 变量名是否存在拼写错误或书写错误。
- 数据类型是否一致。

除局部数据结构外，还应尽可能对影响被测单元的全局数据结构进行检查。

3. 独立路径

对被测单元的基本路径集合中的独立路径进行测试，以及对循环结构进行测试，可以发现大量的路径错误，例如不正确的计算，不正确的比较和不正常的控制流等。

不正确的计算主要包括：

- 运算的优先次序不正确，或误解了运算的优先次序；
- 运算的方式错误，例如运算的对象在类型上不相容、算法错误、初始化错误、运算精度不够、表达式的符号不正确等。

比较和控制流的错误主要包括：

- 进行比较的两个数类型不同；
- 不正确的逻辑运算符或优先次序；
- 因浮点运算精度问题而造成的两值不等；
- 关系表达式中的变量和关系运算符错误；
- 循环次数多一次或少一次；
- 循环终止条件错误；
- 当遇到发散的迭代时无法终止的循环；
- 对循环变量的修改不恰当。

4. 出错处理

被测单元不仅能处理常规的输入，对于出错的情况，也应能事先预见并进行适当的处理，这样的被测单元的功能才是完善的。被测单元的出错处理功能的缺陷主要包括：

- 出错的描述难以理解；
- 出错的描述不足以定位错误和确定出错原因；
- 显示的错误与实际的错误不符；
- 对出错条件的处理不正确；
- 在对错误进行处理之前，出错条件已引起系统的干预。

5. 边界条件

发生在被测单元边界上的错误主要有：

- 对循环体执行 0 次或最大次数时发生的错误；
- 某变量取最大值或最小值时发生的错误；
- 数据流、控制流中恰好等于、大于、小于确定的比较值时发生的错误。

5.4.2　单元测试用例的设计思路

在单元测试阶段，除进行代码检查等静态测试活动以外，均需要设计测试用例并执行它们，以验证被测单元是否达到了设计规格说明的预期要求。设计测试用例的依据主要是软件的详细设计文档和被测单元的源代码清单。

单元测试用例设计步骤如下。

（1）为被测单元运行设计测试用例

设计测试用例，使被测单元可以在建立的单元测试环境中运行。

（2）为正向测试设计测试用例

设计测试用例，验证被测单元能否实现设计文档中规定的各项功能。

（3）为逆向测试设计测试用例

设计测试用例，验证在输入数据不符合要求的情况下被测单元能否进行恰当的处理。

（4）为满足特殊需求设计测试用例

设计测试用例，验证被测单元能否满足某特殊的要求，例如在安全性、保密性方面的要求。

（5）为代码覆盖设计测试用例

设计测试用例，以达到特定的覆盖标准。为达到特定的覆盖标准，仅仅在设计测试用例前提出覆盖标准往往是不够的，还需要在执行测试用例后对覆盖率进行分析，判断是否达到了特定的覆盖标准，进而补充测试用例。

在单元测试阶段的动态测试活动中，白盒及黑盒测试方法测试用例的使用孰先孰后？一般说来，由于黑盒测试是从被测单元外部进行测试，成本较低，可先对被测单元进行黑盒测试，即验证其能否完成规格说明中指定的功能，之后再进行白盒测试，即统计黑盒测试用例的覆盖率，判断测试是否达到了指定的覆盖标准，针对未覆盖的逻辑单元设计测试用例，以弥补黑盒测试不彻底的不足。

5.5 单元测试工具 CppUnit 简介

在本章的前几节中，我们对单元测试进行了介绍，下面结合当前广泛使用的单元测试工具 CppUnit 介绍单元测试的执行过程。

5.5.1 CppUnit 简介

CppUnit 是 XUnit 家族的一员。XUnit 家族成员有很多，如 JUnit，CppUnit，PythonUnit 等。CppUnit 的最初版本移植自 JUnit，是一个非常优秀的专门面向 C++ 的开源的单元测试框架。

CppUnit 是一个基于测试驱动开发的测试框架，它为我们进行测试驱动开发提供了一个有力的工具，借助它可以进行快速的单元测试。

下面主要介绍 VC 6.0 环境下 CppUnit 的使用方法。

1. CppUnit 的安装和环境配置

（1）首先下载 CppUnit。CppUnit 为压缩文件，以下使用的版本是 CppUnit 1.12.0。

（2）编译 CppUnit 工程文件。

① 将 CppUnit1.12.0.tar.gz 解压缩到本地硬盘，如 E:\vc prj\cppunit1.12.0，如图 5-3 所示。

② 启动 VC 6.0，选择 File 菜单，单击 Open Workspace 选项，在打开的 Open Workspace 对话框中，选择 cppunit1.12.0\src 文件夹中的工程文件 CppUnitLibraries.



图 5-3　CppUnit1.12.0.tar.gz 文件解压缩

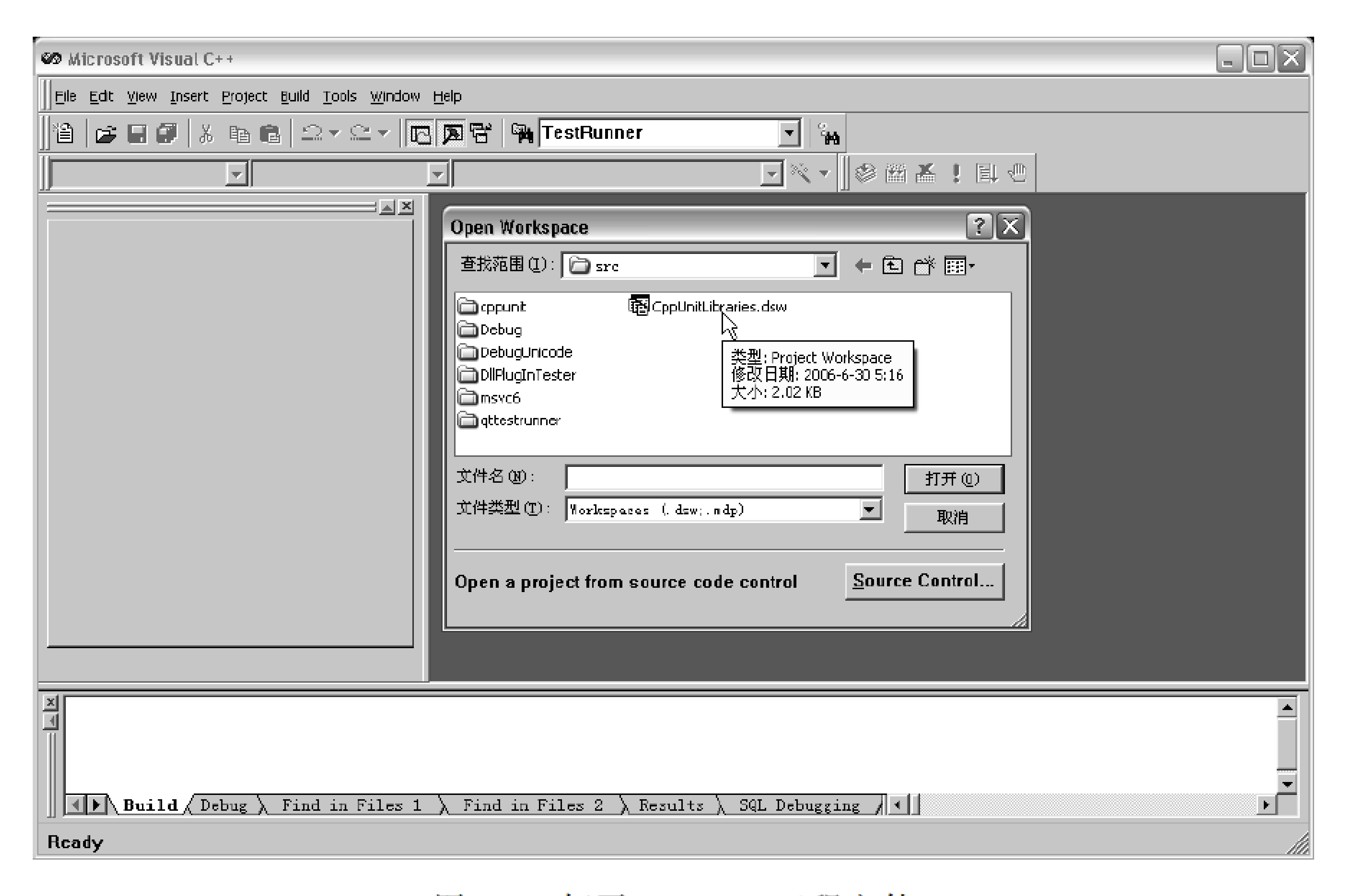


图 5-4　打开 CppUnit 工程文件

dsw，单击“打开”按钮，如图 5-4 所示。

③ 选中 TestPlugInRunner 为当前工程，选择 Build 菜单的 Batch Build 选项，打开 Batch Build 对话框，如图 5-5 所示。选中所有 Project，然后单击 Build 按钮，将生成 CppUnit 的库文件，其位置在 cppunit1.12.0\lib 目录下。对于 CppUnit_dll 项目，其 Release 版生成 CppUnit_dll.lib 和 CppUnit_dll.dll；Debug 版生成 CppUnitd_dll.lib 和

CppUnitd_dll.dll。这是 CppUnit 基本类库。对于 TestRunner 项目,其 Release 版生成 TestRunner.lib 和 TestRunner.dll,Debug 版生成 TestRunnerd.lib 和 TestRunnerd.dll。这是使用 MFC 的图形化界面的类库。

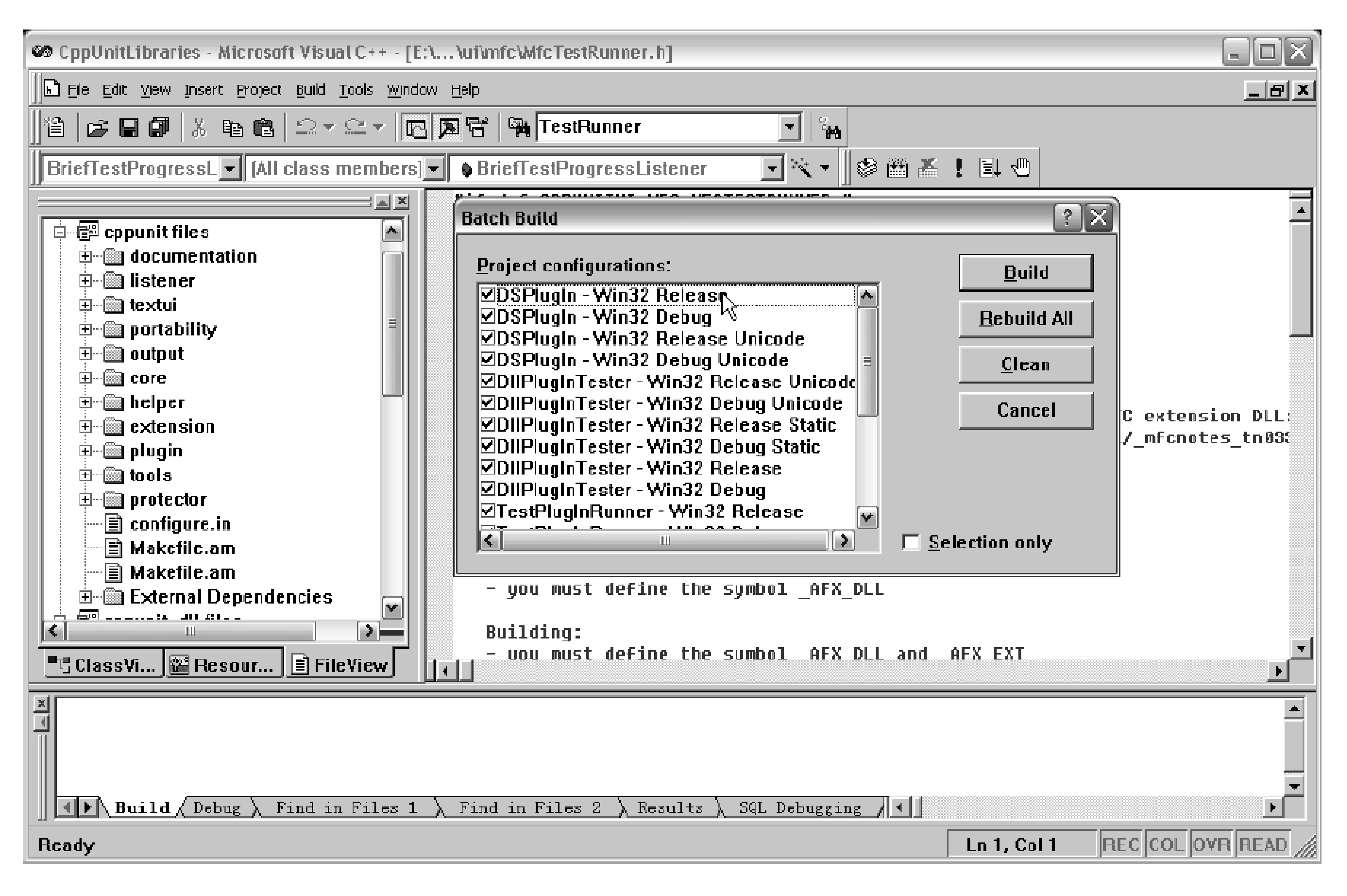


图 5-5 Build CppUnit 工程文件

(3) 设置 VC 6.0 环境。

① 选择 Tools 菜单,单击 Options 选项,如图 5-6 所示。

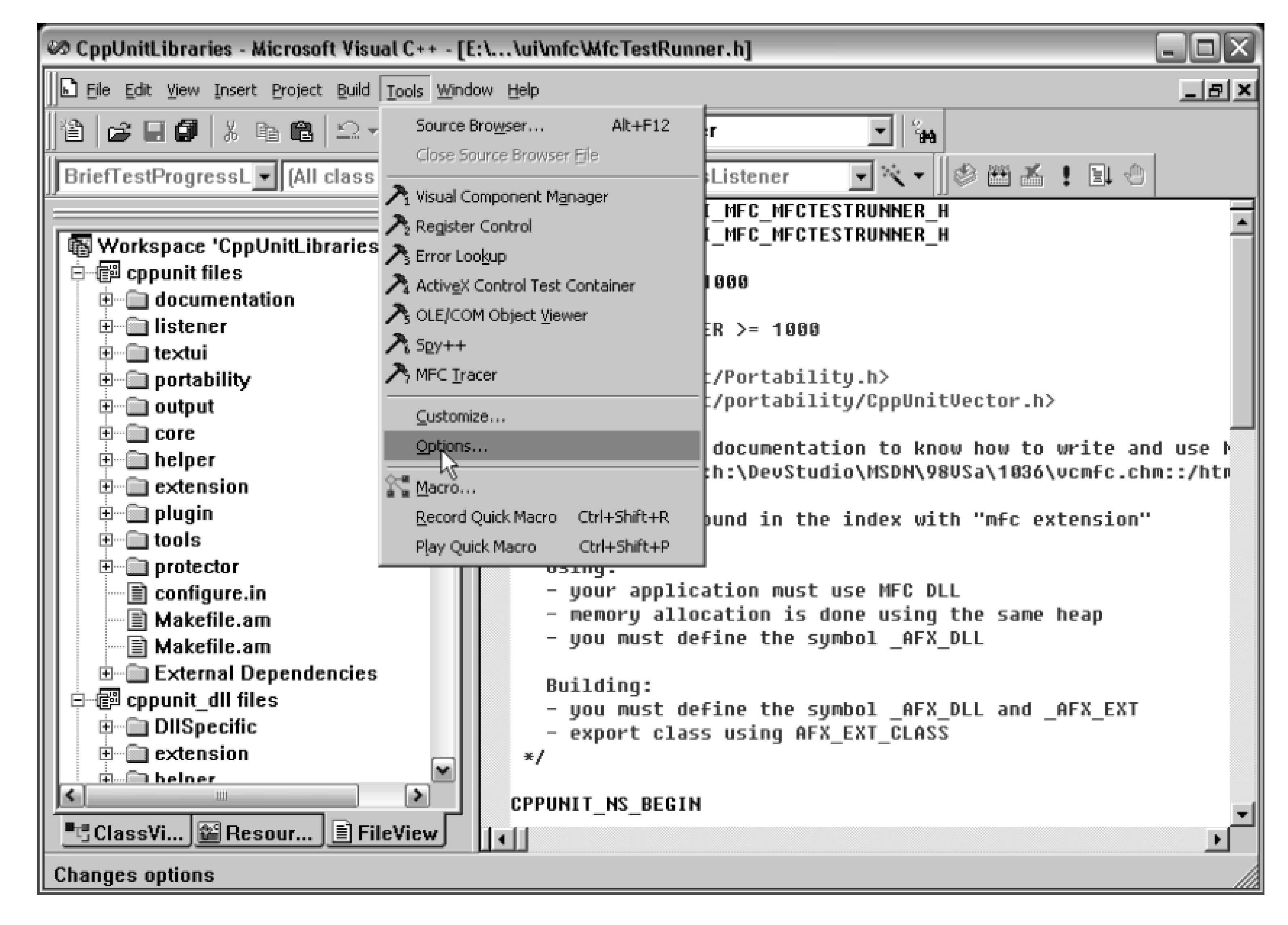


图 5-6 Options 菜单项

打开 Options 对话框，选择 Directories 选项卡，在 Show directories for 下拉列表框中选择 Include files 选项，在 Directories 框中添加路径 E:\vc prj\CPPUnit1.12.0\include，如图 5-7 所示。

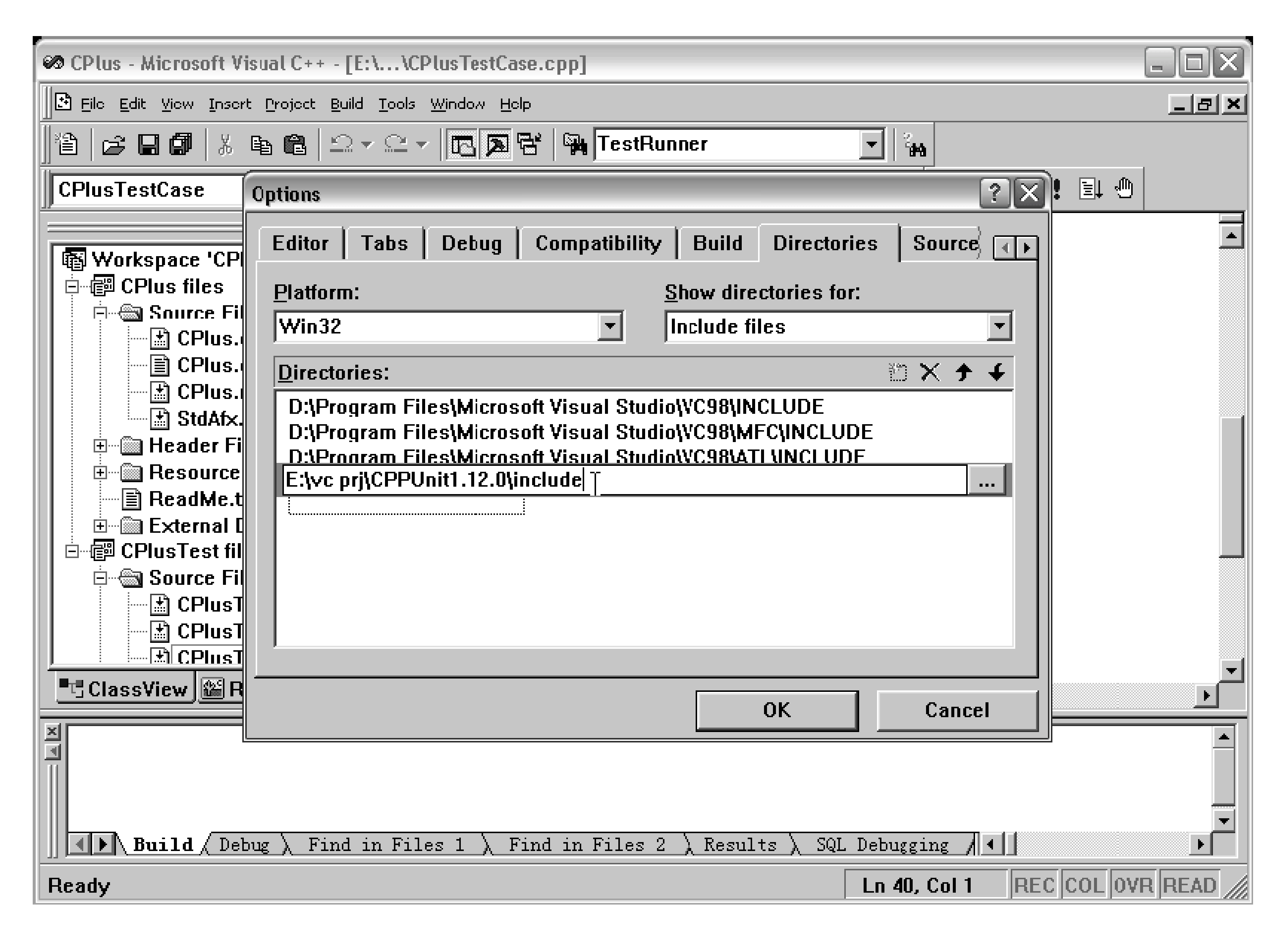


图 5-7　设置 Include files 路径

使用同样方法，添加 Library files 路径“E:\vc prj\cppunit1.12.0\lib”。

② 选择 Project 菜单，单击 Settings 选项（注意应先打开一个项目），打开 Project Settings 对话框，选择 Link 选项卡，在 Category 下拉列表框中选择 General 选项，在 Object/Library modules 文本框中填入 testrunnerd.lib cppunitd.lib，如图 5-8 所示。

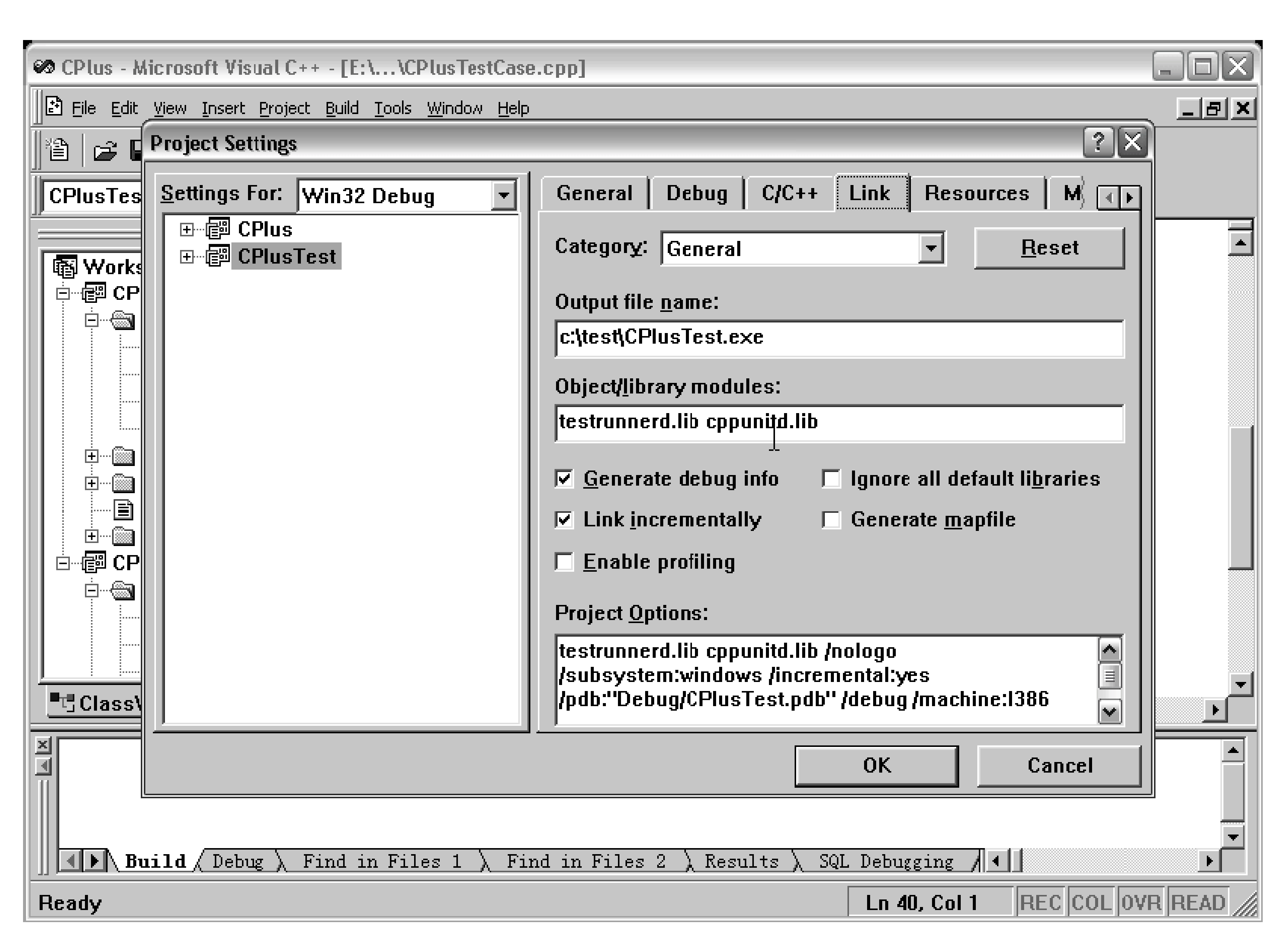


图 5-8　设置 Link 选项

选择 C/C++ 选项卡，在 Category 下拉列表框中选择 Code Generation 选项，在 Use run-time library 下拉列表框中选择 Debug Multithreaded DLL 选项，如图 5-9 所示。

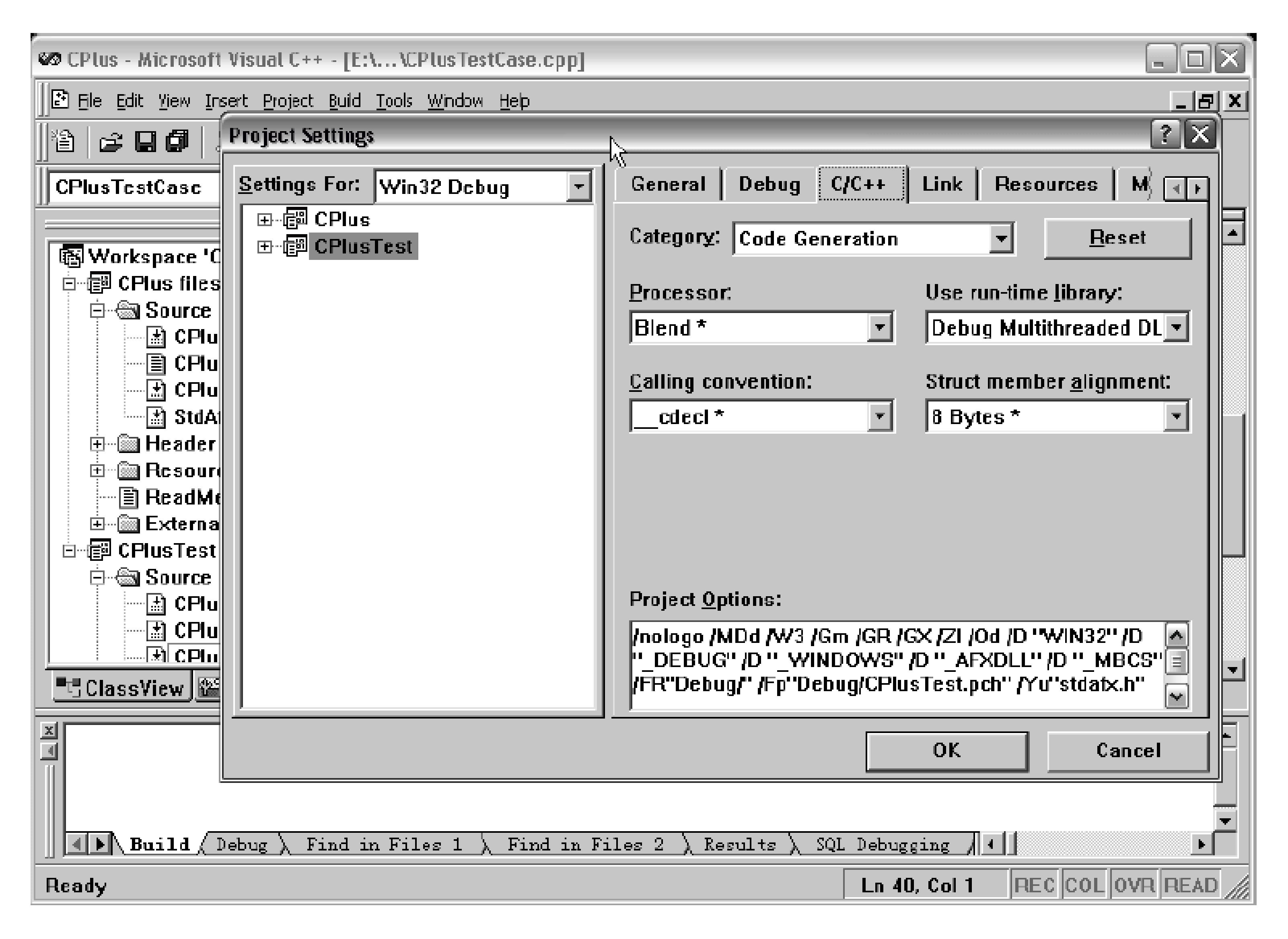


图 5-9 设置 run-time library

③ 在系统的环境变量 Path 中添加相应路径。该步骤是为了确保在以后的测试项目中，系统能够找到相应的动态链接库，例如 TestRunnerd.dll。

在 Windows 桌面上选中“我的电脑”图标，单击右键，在弹出的快捷菜单中选择“属性”选项，打开“系统属性”对话框，选择“高级”选项卡，如图 5-10 所示。

单击“环境变量”按钮，出现“环境变量”对话框，在“系统变量”选项区中选择 Path 系统变量，单击“编辑”按钮，如图 5-11 所示。



图 5-10 环境变量设置



图 5-11 设置 Path

打开“编辑系统变量”对话框，在 Path 系统“变量值”文本框中添加 E:\vc prj\cppunit1.12.0\include 和 E:\vc prj\cppunit1.12.0\lib，变量值之间用分号“;”分隔，单击“确定”按钮，如图 5-12 所示。



图 5-12　添加 path 值

至此，CppUnit 的环境就配置完成。

2. CppUnit 基本原理

在 CppUnit 中，一个或一组测试用例的测试对象被称为 Fixture。Fixture 就是被测试的目标，可以是一个对象或者一组相关的对象，甚至一个函数。有了被测试的 Fixture，就可以对这个 Fixture 的某个功能、某个可能出错的流程编写测试代码。

使用 CppUnit 的步骤如下。

(1) 生成 CppUnit::TestFixture 或 CppUnit::TestCase 的一个派生类（形如 C×××TestCase），该类我们称其为测试类。在该类中为测试用例添加一个成员函数（形如 test×××）用于测试目标代码。可以在该类中实现父类中的虚函数 setUp()和 tearDown()，其中 setUp()函数完成代码初始化工作，tearDown()函数完成清理工作。注意 setUp()和 tearDown()的实现不是必须的。

(2) 在 C×××TestCase 类中添加如下宏：

```
CPPUNIT_TEST_SUITE(C×××TestCase)
CPPUNIT_TEST(test×××)
CPPUNIT_TEST_SUITE_END()
```

这三个宏非常关键，通过这三个宏，就把 C×××TestCase 类和 test×××成员函数注册到测试列表中了。

(3) 添加宏：

```
CPPUNIT_TEST_SUITE_REGISTRATION(C×××TestCase)
```

注意，该宏位于所有类和函数之外。该宏将 C×××TestCase 测试类注册到 TestFactory 工厂中，在整个基于 CppUnit 的程序框架中，该宏起到一个桥梁的作用。

(4) 在 test×××成员函数中添加宏：

```
CPPUNIT_ASSERT_EQUAL(value, function)
```

其中，function 为被测函数，value 为预期输出值。此宏称为断言宏，如果 function 的输出与 value 值不符，程序将返回一个错误信息，表示在测试目标代码中发现了一个错误。在 test×××成员函数中可以添加多个断言宏。使用这种机制，实际上就可以将多个测试用例添加到测试程序中。有关断言宏，读者可以查看 TestAssert.h 文件。

(5) 在主函数（如 main()函数）或程序的初始化函数（如基于 MFC 对话框程序的 initInstance()函数）中添加测试代码。主要有下面 3 条语句：

① CppUnit::TextUi::TestRunner。该语句生成 TestRunner 的对象。目前，CppUnit 提供了 3 种 TestRunner，包括：

```
CppUnit::TextUi::TestRunner          // 文本方式的 TestRunner
CppUnit::QtUi::TestRunner            // QT 方式的 TestRunner
CppUnit::MfcUi::TestRunner           // MFC 方式的 TestRunner
```

② runner.addTest(suite)。该语句添加测试对象到 TestRunner。其中，suite 是一个 CppUnit::Test * 类型。一般 suite 的生成方法为

```
CppUnit::Test * suite = CppUnit::TestFactoryRegistry::getRegistry().makeTest();
```

其中，CppUnit :: TestFactoryRegistry :: getRegistry ( ) 返回一个 CppUnit :: TestFactoryRegistry & 类型。

③ runner.run()。该语句运行测试对象。

5.5.2 CppUnit 单元测试实例 1

1. 问题描述

对求 3 个整数中最大值的程序进行测试。

2. 测试用例

对求 3 个整数中最大值的程序进行测试，应先求出 3 个整数之间关系的等价类表，如表 5-1 所示。表 5-2 是根据表 5-1 设计的测试用例。

表 5-1 3 个整数之间关系的等价类表

编号	等价类	预期输出	编号	等价类	预期输出	编号	等价类	预期输出
1	a>b>c	a	6	c>b>a	c	11	b=c>a	b 或 c
2	a>c>b	a	7	a=b>c	a 或 b	12	a>b=c	a
3	b>a>c	b	8	c>a=b	c	13	a=b=c	a 或 b 或 c
4	b>c>a	b	9	a=c>b	a 或 c			
5	c>a>b	c	10	b>a=c	b			

表 5-2 测试用例

编号	[a, b, c]	预期输出	编号	[a, b, c]	预期输出	编号	[a,b,c]	预期输出
1	[9, 8, 7]	9	6	[7, 8, 9]	9	11	[8, 9, 9]	9
2	[9, 7, 8]	9	7	[9, 9, 8]	9	12	[9, 8, 8]	9
3	[8, 9, 7]	9	8	[8, 8, 9]	9	13	[9, 9, 9]	9
4	[7, 9, 8]	9	9	[9, 8, 9]	9			
5	[8, 7, 9]	9	10	[8, 9, 8]	9			

3. 使用 CppUnit 进行单元测试

在求 3 个整数中最大数的程序中，主要对类 CMaxNum 中的 getMaxNum(int *,

int)函数进行测试。以下给出源代码。

```
// CMax. cpp
class CMaxNum
{
    public:
    CMaxNum(){};
    ~CMaxNum(){};
    int getMaxNum(int * , int);
};
// CMax. cpp
#include <iostream. h>
#include "CMax. h"
int CMaxNum::getMaxNum(int *  a, int n)
{
    int MaxNum;
    MaxNum=a[0];
    for(int i=1;i<n-1; i++ )                     //该处存在错误
    {
        f(MaxNum<a[i])
        MaxNum=a[i];
    }
    return MaxNum;
}
// MaxTestCase. cpp
#include "Cmax. h"
#include "CppUnit/TestFixture. h"
#include "CppUnit/extensions/HelperMacros. h"
// 定义测试类
class CMaxTestCase : public CppUnit::TestFixture
{
    CPPUNIT_TEST_SUITE(CMaxTestCase);     //创建一个 TestSuite
    CPPUNIT_TEST(testMaxNum);             //添加一个 TestCase
    CPPUNIT_TEST_SUITE_END();             //结束一个 TestSuite 创建
    public:
    //添加测试用例
    void testMaxNum()
    {
        CMaxNum num;
        int a[3]={9, 8, 7};                                   //第 1 个测试用例
        CPPUNIT_ASSERT_EQUAL(9, num. getMaxNum(a, 3));        //添加断言测试
        a[0]=9; a[1]=7; a[2]=8;                               //第 2 个测试用例
        CPPUNIT_ASSERT_EQUAL(9, num. getMaxNum(a, 3));        //添加断言测试
        a[0]=8; a[1]=9; a[2]=7;                               //第 3 个测试用例
        CPPUNIT_ASSERT_EQUAL(9, num. getMaxNum(a, 3));        //添加断言测试
    }
};
//自动注册测试类
CPPUNIT_TEST_SUITE_REGISTRATION(CMaxTestCase);
// MaxApp. cpp
```

```
#include <CppUnit/ui/text/TestRunner.h>
#include <CppUnit/extensions/TestFactoryRegistry.h>
void main()
{
    CppUnit::Test * suite = CppUnit::TestFactoryRegistry::getRegistry().makeTest();
    //定义运行实例
    CppUnit::TextUi::TestRunner runner;
    //添加测试,将测试包注册到匿名的 TestFactory 工厂中
    runner.addTest(suite);
    //运行测试
    runner.run();
}
```

先添加 4 个测试用例,程序运行出现错误,结果如图 5-13 所示。

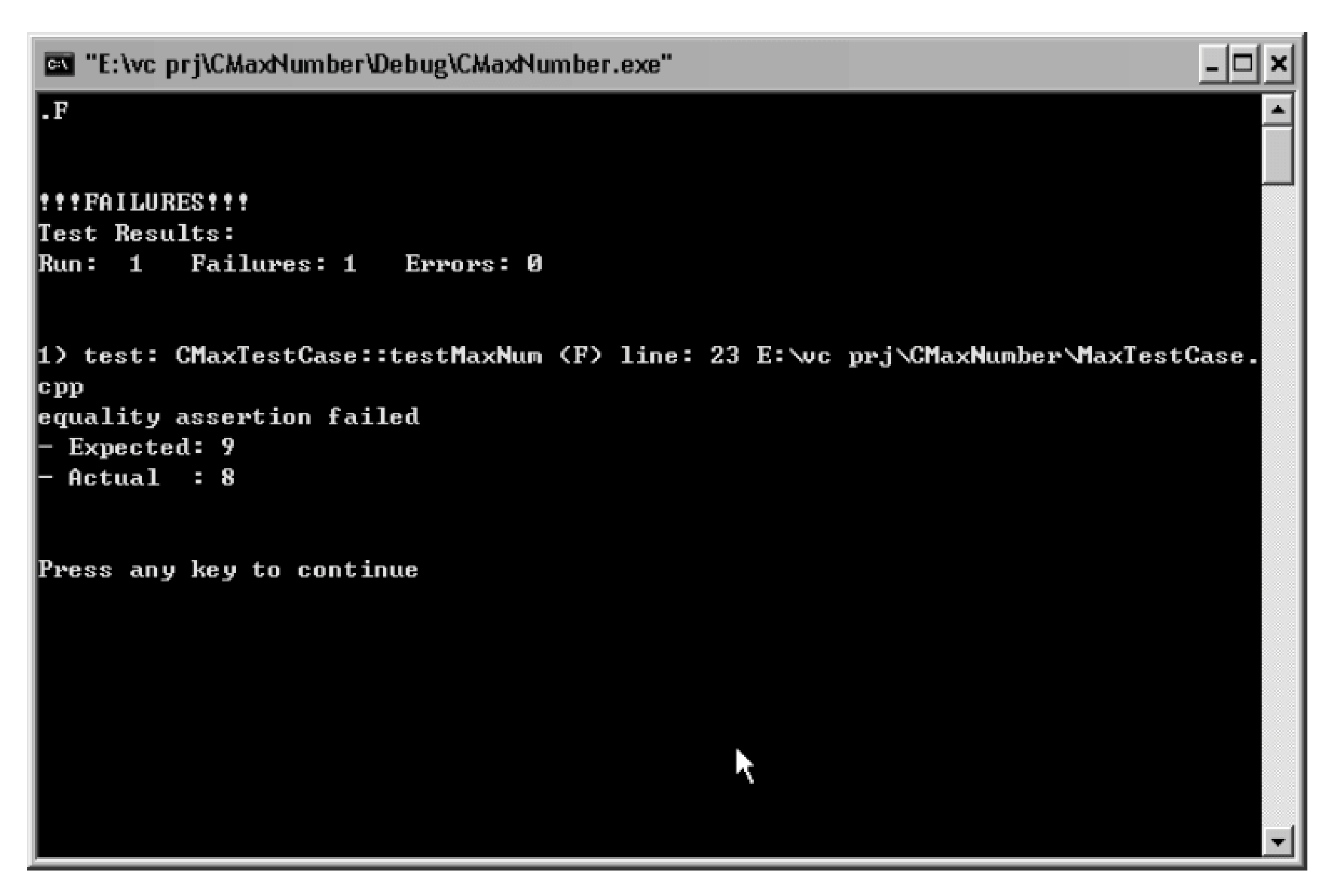


图 5-13 运行结果显示

程序运行结果明确指出了错误的位置在 MaxTestCase.cpp 的 23 行,错误的原因是:程序求得的最大值是 8,实际值(预期值)应该是 9。经过检查,发现了程序的错误在 CMax.cpp 中的 for(int i=1;i<n-1;i++)语句,应将此语句改为 for(int i=1;i<n;i++)。经过修改后,程序的运行结果正确,如图 5-14 所示。

排除错误后,可以接着将后 8 个测试用例添加到程序中并运行,直到测试全部通过。通过该例子我们对 CppUnit 的使用应该有了一个基本的了解。在 5.5.3 小节将给出一个复杂一些的例子,并使用基于 MFC 界面方式的输出测试结果。

5.5.3 CppUnit 单元测试实例 2

1. **问题描述**

构造斐波那契(Fibonacci)数列,用测试驱动开发的方法完成类 CFib(类 CFib 用于构造斐波那契数列)。

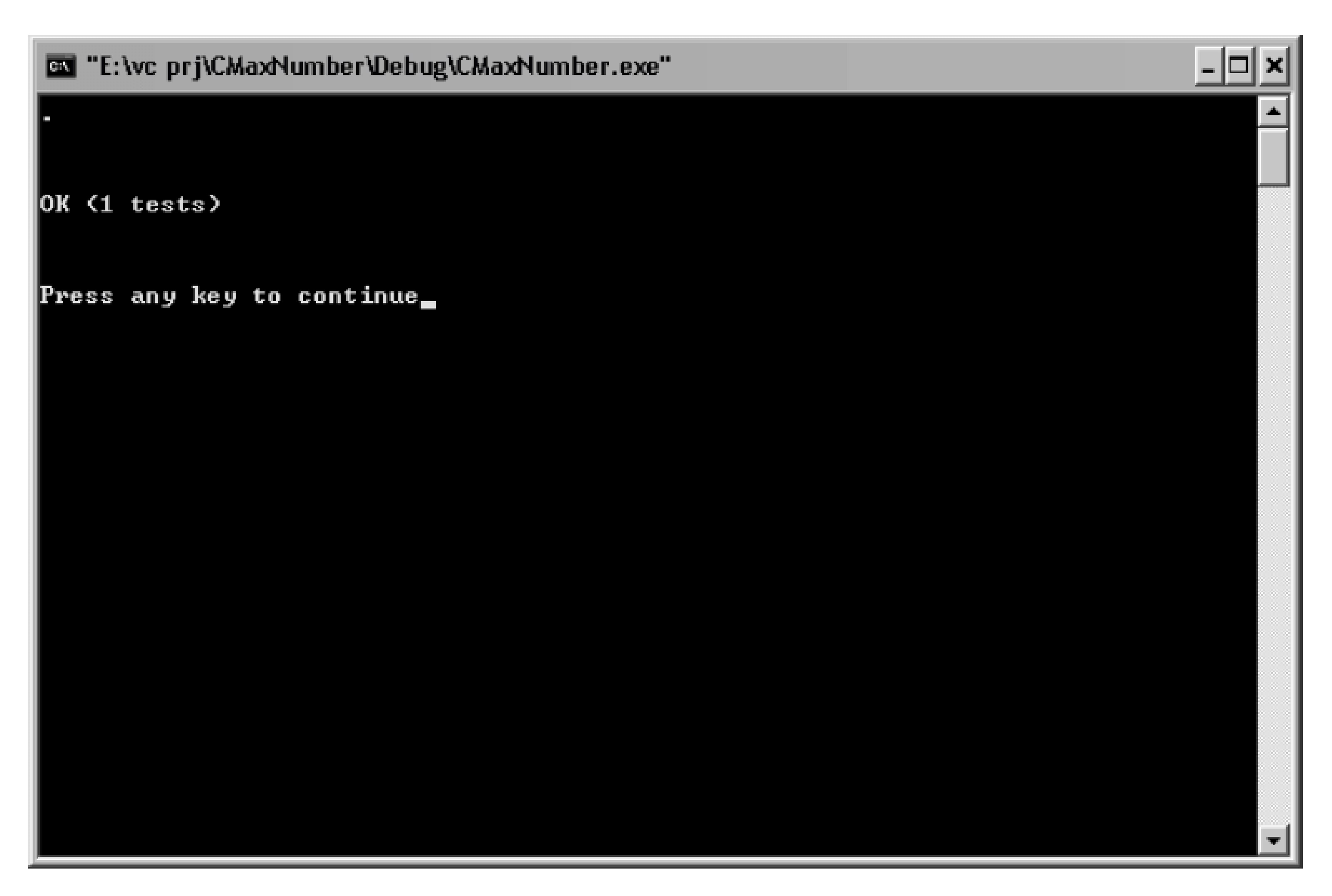


图 5-14　修改语句后的运行结果

2. 使用 CppUnit 完成测试驱动开发

(1) 创建基于对话框的应用程序。本实例测试用例结果以 GUI 方式输出，由于 GUI 方式输出需要使用 MFC，所以我们首先在 VC 中创建一个基于对话框的应用程序 FibTestApp，如图 5-15 所示。

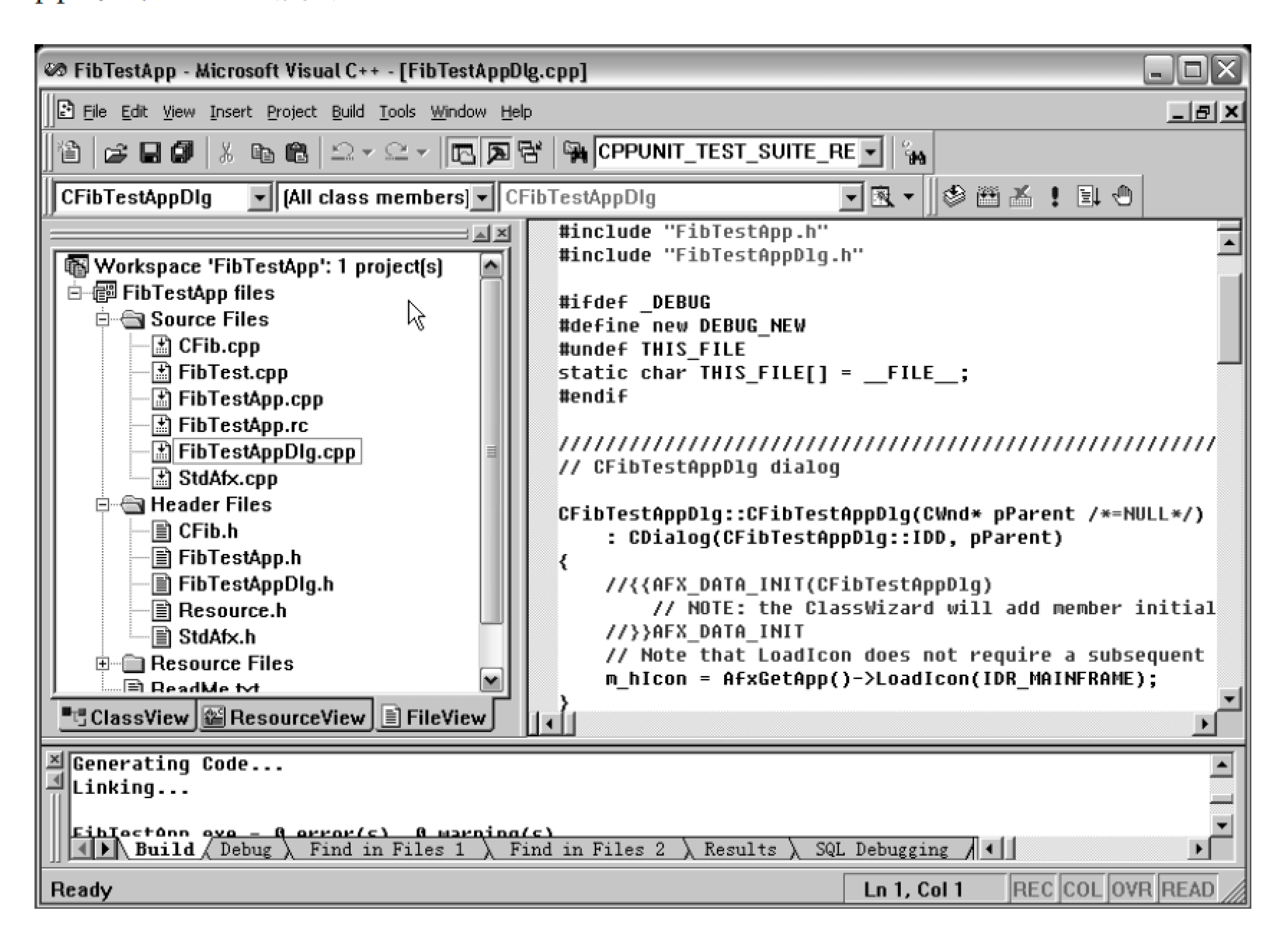


图 5-15　创建对话框程序

本例使用 CppUnit 进行单元测试的方法与上一个例子基本相同，不同点在于：基于字符界面的 CppUnit 测试程序(如上例)，其生成运行实例的代码是在 main()函数中实现的。由于 MFC 对代码进行了封装，所以对基于字符界面的 CppUnit 测试程序(如本例)，

我们将生成运行实例的代码写在 InitInstance()中，同时注释掉生成对话框的代码。其代码如下：

```
// FibTestApp.cpp
#include "FibTestApp.h"
#include "FibTestAppDlg.h"
#include <CppUnit/ui/mfc/TestRunner.h>
#include <CppUnit/extensions/TestFactoryRegistry.h>
BOOL CFibTestAppApp::InitInstance()
{
    #ifdef _AFXDLL
    Enable3dControls();            //Call this when using MFC in a shared DLL
    #else
    Enable3dControlsStatic();      //Call this when linking to MFC statically
    #endif

    CppUnit::Test * suite = CppUnit::TestFactoryRegistry::getRegistry().makeTest();
    //定义运行实例
    CppUnit::MfcUi::TestRunner runner;
    //添加测试，将测试包注册到匿名的 TestFactory 中
    runner.addTest(suite);
    //运行测试
    runner.run();

/*    CFibTestAppDlg dlg;
      m_pMainWnd = &dlg;
      int nResponse = dlg.DoModal();
      if (nResponse == IDOK)
      {
          //TODO: Place code here to handle when the dialog is
          //dismissed with OK
      }
      else if (nResponse == IDCANCEL)
      {
          //TODO: Place code here to handle when the dialog is
          //dismissed with Cancel
      } */
      return FALSE;
}
```

(2) 定义测试类。

```
// FibTest.cpp:
#include "CFib.h"
#include "CppUnit/TestFixture.h"
#include "CppUnit/extensions/HelperMacros.h"
//定义测试类 FibTest
class CFibTest:public CppUnit::TestFixture
{
    CPPUNIT_TEST_SUITE(CFibTest);      //定义测试包
```

```
    CPPUNIT_TEST(testFib);                    //添加测试用例 testFib
    CPPUNIT_TEST_SUITE_END();                 //结束测试包定义
    public:
    //测试用例
    void testFib()
    {
        CFib fb;
        //在此添加断言测试
        CPPUNIT_ASSERT_EQUAL(1, fb.fib(1));
        CPPUNIT_ASSERT_EQUAL(1, fb.fib(2));
        //添加断言测试第 3 个元素
        CPPUNIT_ASSERT_EQUAL (2, fb.fib(3));
        //添加断言测试第 5 个元素
        CPPUNIT_ASSERT_EQUAL (5, fb.fib(5));
    }
};
//自动注册测试包
CPPUNIT_TEST_SUITE_REGISTRATION(CFibTest) ;

//CFib.h 被测试类,其成员函数 int fib(int)测试对象
class CFib
{
    public:
    CFib(){}
    ~CFib(){}
    int fib(int);
};
//CFib.cpp :
#include "CFib.h"
int CFib::fib(int n)
{
    return 1;
}
```

(3) 首先编写一个测试用例 testFib,断言 fib(1) = 1,fib(2) = 1。这表示该数列的第 1 个元素和第 2 个元素都是 1。测试用例 testFib 已在前面申明。在此主要是新增 TestFib 测试用例实现,测试第 1 个元素和第 2 个元素。

```
void CFibTest::testFib()
{
    CFib fb;
    //在此添加断言测试
    CPPUNIT_ASSERT_EQUAL(1, fb.fib(1));
    CPPUNIT_ASSERT_EQUAL(1, fb.fib(2));
}
```

(4) 测试用例不能编译通过。为了让测试用例通过,需实现 fib 函数(被测函数)。

```
int CFib::fib(int n)
{
```

```
    return 1;
}
```

(5) 编译后测试用例可以运行。继续测试第 3 个元素。

```
void CFibTest::testFib()
{
    CFib fb;
    //在此添加断言测试
    CPPUNIT_ASSERT_EQUAL(1, fb.fib(1));
    CPPUNIT_ASSERT_EQUAL(1, fb.fib(2));
    //添加断言测试第 3 个元素
    CPPUNIT_ASSERT_EQUAL (fb.fib(1)+ fb.fib(2), fb.fib(3));
}
```

(6) 新增加的断言导致执行测试用例失败,如图 5-16 所示。

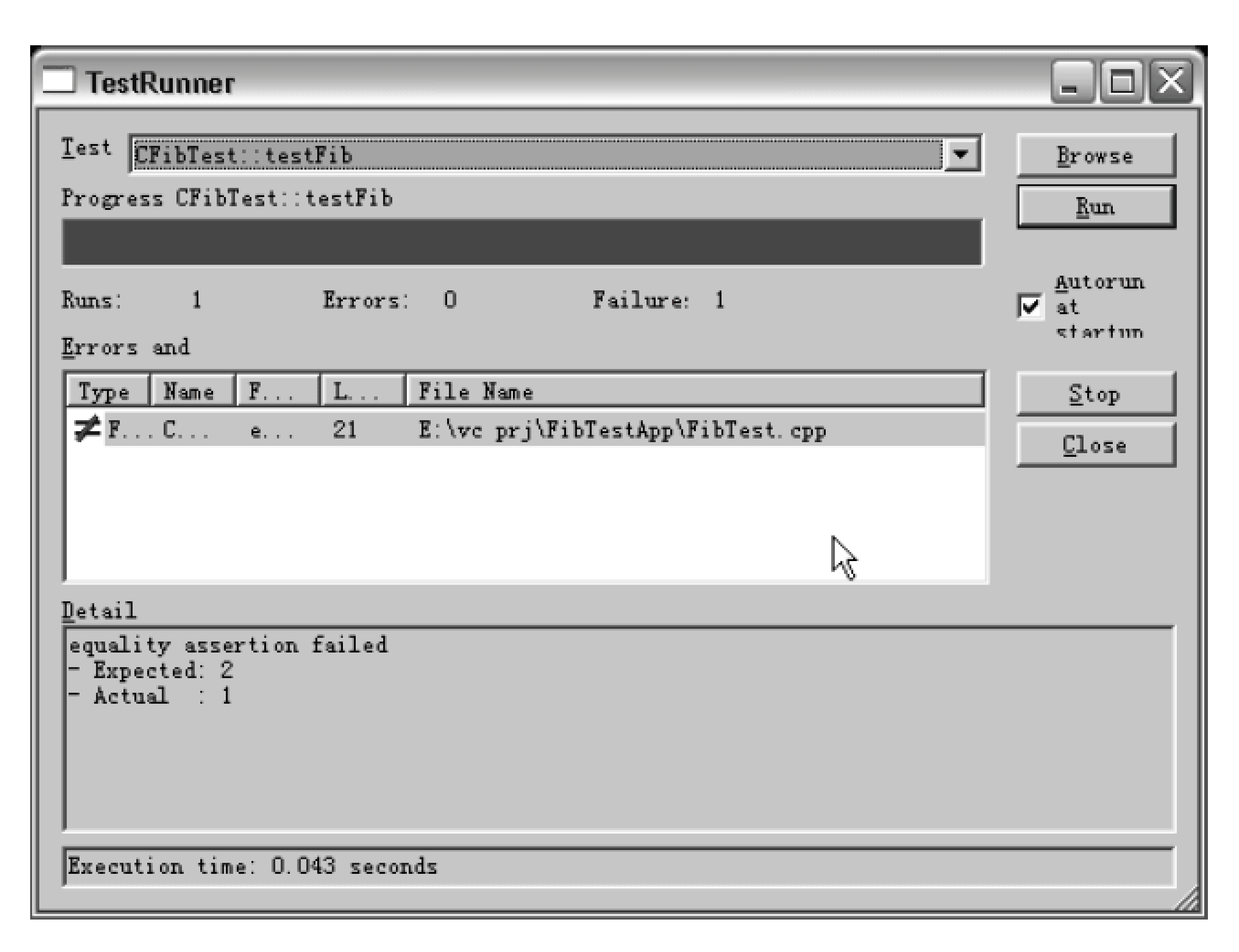


图 5-16 测试运行结果(1)

需对 fib 函数进行修改如下。

```
int CFib::fib(int n)
{
    if (n == 3) return fib(1)+fib(2);
    return 1;
}
```

(7) 测试用例运行通过,如图 5-17 所示。

继续测试第 4 个元素。

```
void CFib::testFib()
{
    CFib fb;
    //在此添加断言测试
    CPPUNIT_ASSERT_EQUAL(1, fb.fib(1));
    CPPUNIT_ASSERT_EQUAL(1, fb.fib(2));
```

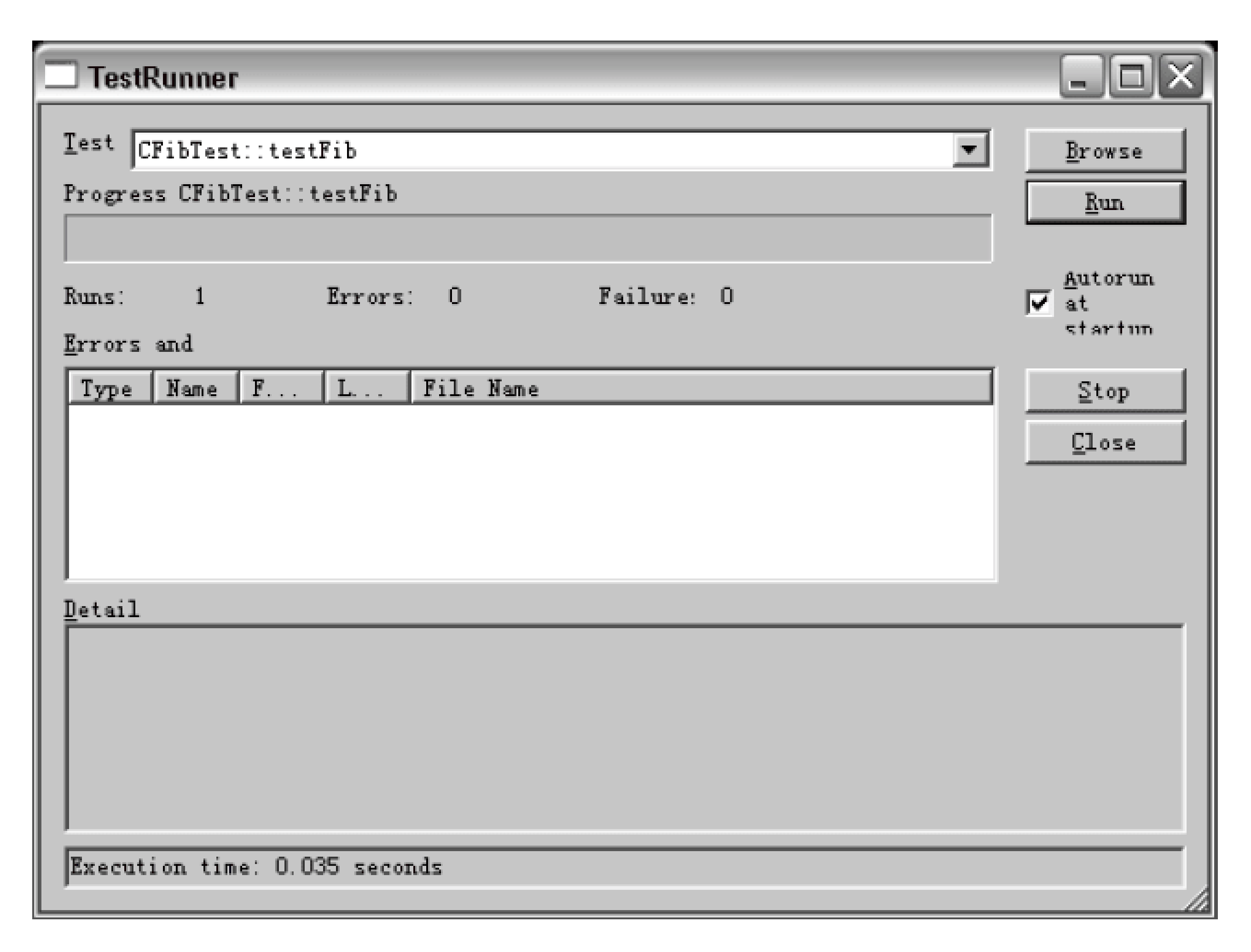


图 5-17　测试运行结果(2)

```
    //添加断言测试第 3 个元素
    CPPUNIT_ASSERT_EQUAL(fb.fib(1)+ fb.fib(2), fb.fib(3));
    //添加断言测试第 4 个元素
    CPPUNIT_ASSERT_EQUAL(fb.fib(2)+ fb.fib(3), fb.fib(4));
}
```

(8) 新增加断言又导致测试用例执行失败,因此继续修改 fib 函数。

```
int CFib::fib(int n)
{
    if (n == 3) return fib(1)+fib(2);
    if (n == 4) return fib(2)+fib(3);
    return 1;
}
```

(9) 测试用例可运行,但是代码出现了冗余,所以需要对 fib 函数进行重构。

```
int CFib::fib(int n)
{
    if (n == 1 || n == 2) return 1;
    else return fib(n - 1) + fib(n - 2);
}
```

(10) 由于还未考虑错误输入值(如输入值 0,-1 等),故添加错误输入值测试项。

```
void CFib::testFib()
{
    CFib fb;
    //在此添加断言测试
    CPPUNIT_ASSERT_EQUAL(1, fb.fib(1));
    CPPUNIT_ASSERT_EQUAL(1, fb.fib(2));
    //添加断言测试第 3 个元素
    CPPUNIT_ASSERT_EQUAL(fb.fib(1)+ fb.fib(2), fb.fib(3));
    //添加断言测试第 4 个元素
```

```
    CPPUNIT_ASSERT_EQUAL(fb.fib(2)+ fb.fib(3), fb.fib(4));
    //添加错误输入值测试项
    CPPUNIT_ASSERT_EQUAL(0, fb.fib(0));
    CPPUNIT_ASSERT_EQUAL(0, fb.fib(-1));
}
```

(11) 再次运行测试用例,无法通过,因此还需修改 fib 函数如下。

```
int CFib::fib(int n)
{
    if (n <= 0) return 0;
    if (n == 1 || n == 2) return 1;
    else return fib(n - 1) + fib(n - 2);
}
```

(12) 再次运行所有测试用例,所有测试用例运行通过,如图 5-18 所示。

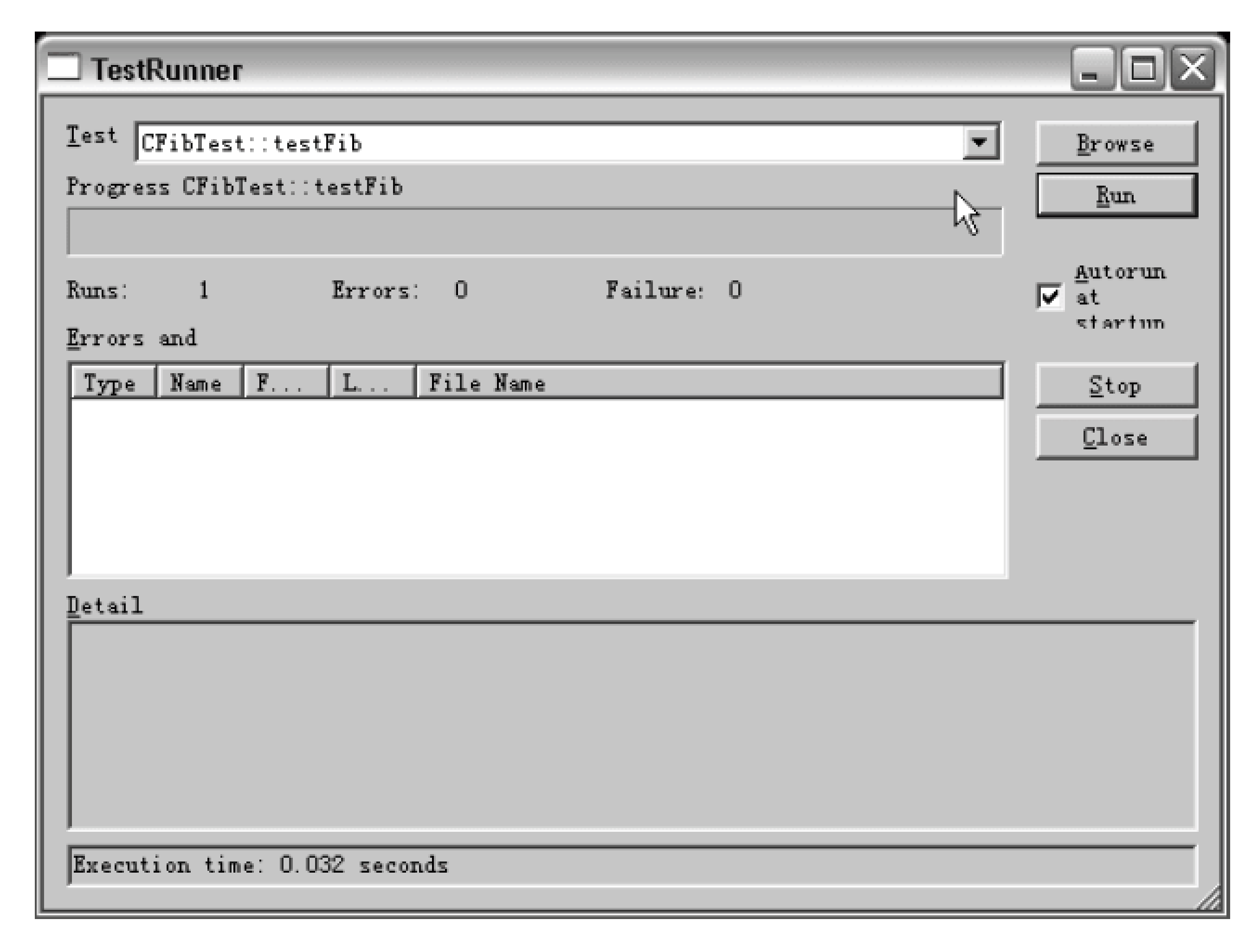


图 5-18　测试运行结果(3)

至此,使用 CppUnit 对斐波那契(Fibonacci)数列进行测试驱动开发的全过程完毕。

5.6　小结

单元测试是所有测试活动中最早进行的一种测试,也是极为重要的一种测试。它针对软件的基本组成单元进行测试,是其他测试活动的基础。它看似烦琐,却能以最低的成本发现和修复软件单元中的错误。忽视单元测试是十分不明智的做法。

应有计划地执行单元测试,并在整个软件开发的周期内对其进行维护,使单元测试可重用。

单元测试采用白盒及黑盒测试方法对被测单元从静态和动态两方面进行测试,以验证编码能否达到设计规格说明的要求。

单元测试中的被测单元往往不是一个可以独立运行的程序,故在执行单元测试阶段的动态测试时,应注重建立单元测试的环境,以达到对被测单元进行测试的目的。

目前,有很多单元测试工具及框架可供选择,这有助于提高单元测试的自动化程度,且大都支持测试驱动开发,如本章介绍的 CppUnit。

习　　题

1. 单元测试的含义是什么?“单元”的概念在任何开发场合是一成不变的吗?试谈谈你的观点。

2. 单元测试的依据是什么?进行单元测试的目的何在?

3. 单元测试的优点有哪些?

4. 简述单元测试的步骤。

5. 驱动模块和桩模块的含义分别是什么?在单元测试中,一定要开发驱动模块和桩模块吗?为什么?

6. 针对某被测程序,分别举一个驱动模块和桩模块的例子。

7. 构造单元测试环境的意义何在?

8. 单元测试应主要从哪些方面对被测单元进行检查?

9. 在单元测试的动态测试活动中,白盒与黑盒测试方法测试用例的使用有什么样的关系?试谈谈你的观点。

第 6 章 集成测试

本章要点：

- 集成测试的任务。
- 集成测试的对象。
- 接口的分类。
- 集成测试的方法。
- 集成测试的步骤。
- 集成测试的策略。
- 集成测试中模块和接口的确定。
- 关键模块的概念。

对于被测软件，仅进行单元测试是远远不够的，通过了单元测试的软件单元还不能保证在集成过程中不出现错误。要排除在单元集成过程中的各种错误，应进行集成测试。本章介绍集成测试的相关概念和策略。

6.1 集成测试的概念

6.1.1 集成测试的含义

集成测试(Integration Testing)，也称为组装测试、联合测试。在单元测试的基础上，应根据概要设计的要求将各单元组装成子系统或系统，在单元组装过程中，应对单元进行整体测试，发现并清除在单元连接过程中出现的问题，确保集成到一起的各单元能共同完成预期的功能，并达到要求的性能，这就是集成测试的任务。对子系统的集成测试，也称为部件测试。

在集成测试中，回归测试是时常需要使用的。当对集成后的某单元进行了修改，就需要进行回归测试，验证与该单元组装在一起的其他单元(尤其是上层单元)能否正常工作。

集成测试一般由专门的测试小组完成，测试小组由有经验的系统设计人员和编程人员组成，最好是参加了被测项目开发的人员。一般说来，集成测试花费的时间远远超过单元测试。

实践表明，通过了单元测试的单元并不能保证在集成过程中不出现问题，程序在某些局部反映不出来的问题，可能会在全局上暴露出来，影响其功能的实现。在项目实践中，几乎不存在单元在组装过程中不出现任何问题的情况。这是因为单元测试具有不彻底性，它对于单元接口的正确性是无法保障的，以下的问题是在集成测试中必须考虑的。

- 将各模块组装起来的过程中，穿越模块接口的数据是否会丢失。
- 各子功能组合起来，能否达到预期的父功能。
- 某模块的功能是否会对另一模块的功能产生不利影响。
- 全局数据结构是否存在问题。
- 单个模块的误差积累起来，是否会放大到不可接受的程度。

此外，在某些开发模式中，如迭代式开发，设计和实现是迭代进行的。在这种情况下，集成测试的意义还在于它能间接地验证概要设计是否具有可行性。

集成测试的测试对象包括单元间的接口，以及集成后的功能和性能(其中以对功能的测试为主)。

值得一提的是，在如今的测试活动中，单元测试和集成测试的界限逐渐模糊。单元测试阶段也可进行集成，如采用自底向上的方式进行集成，底层的单元先开发并先测试，这样做的目的是避免开发桩模块。所以在实际的测试工作中，不应死抠单元测试与集成测试的区别，只要完成了相应的测试活动即可，以免误入歧途。

6.1.2 接口的分类

对接口的测试是集成测试的主要任务，在集成测试阶段，主要考虑软件系统内部的各种接口，可对其进行如下分类。

1. 函数接口

根据函数之间的调用关系确定的接口。

2. 消息接口

消息接口在用面向对象方法开发的系统及嵌入式系统中比较常见。使用消息接口时，软件的模块间并不直接联系，而是通过消息包(遵循接口协议)发生关系，例如整个系统共用一个或多个消息包队列，消息位于队列的头部。由操作系统进行消息包的调度，取出各队列头部的消息，并调用该消息的处理模块，该处理模块的核心通常是一个被动的有限状态机模型，能根据消息内容和自身状态做出反应，能完成状态的迁移并将发往另一个模块的消息放入消息队列。对消息接口的测试一般用工具来模拟进行。

3. 类接口

在使用面向对象方法开发的软件系统中，类接口是基本的接口。对于类接口的测试，传统的集成测试策略难以适用，详见本书第 8 章。

4. 其他接口

主要包括全局变量、配置表、注册信息、中断等。在进行集成测试时，不应忽略这类接口，若忽略或无意中遗漏了对此类接口的测试，有可能会影响集成测试的效果。对这类接口的测试可借助于测试工具进行。

6.1.3 集成测试的测试方法

在集成测试阶段，应将黑盒测试方法与白盒测试方法结合起来使用。使用黑盒测试方法从接口处对子系统或系统进行测试，在其基础上再选择性地使用白盒测试方法对模块内部路径进行测试。在集成测试阶段，使用白盒测试方法的目的不再是为了满足一定的代码覆盖率，而是为了在进行黑盒测试时不遗漏应有的功能点。也可以理解为，这一阶段的白盒测试是为黑盒测试服务的。

集成测试阶段使用的测试方法可理解为灰盒测试方法。下面简单介绍灰盒测试方法。

灰盒测试是一种介于黑盒测试和白盒测试之间的测试策略，它基于程序运行的外部表现同时又结合程序内部逻辑结构来设计测试用例。灰盒测试与黑盒测试一样，关注被测对象的输出对于输入的正确性，即关注其能否完成预期的功能，而它同时也关注被测对象的内部结构，但这种关注不像白盒测试那样详细、完整，而是有选择性的。进行灰盒测试需花费比黑盒测试多20%～40%的时间。

任何项目都会从灰盒测试中获益。不管是手机中使用的各种嵌入式软件，还是Web页面和服务器端的任何脚本，都可以运用这种测试方法。对于重要的项目，尤其是涉及生命或重大财产安全的项目则更适合于使用这种测试方法。

为进行灰盒测试，需要安装源代码，从源代码编译生成的目录中运行子系统或系统，测试人员应熟悉源代码或具备很强的阅读源代码的能力，还应配置针对编程语言的代码覆盖率工具。

设计灰盒测试用例与黑盒测试用例的主要区别在于，灰盒测试用例的粒度超出黑盒测试用例，也就是说灰盒测试用例不仅仅是从被测对象外部进行测试。因为这种测试用例，主要是覆盖需求功能点，每个功能点可能细分为多种情况，而且需求功能点之间还存在组合的情况，这仅仅依靠黑盒测试是难以实现的，通过覆盖代码中的路径则容易实现这一目标，这在很大程度上提高了测试的有效性。

灰盒测试可以说是将白盒测试和黑盒测试策略的优点结合起来，其主要优点如下：

- 能够进行基于需求的覆盖测试(即覆盖功能点)和基于路径的覆盖测试。
- 可深入被测对象的内部，便于错误的定位、分析和解决。
- 能够保证设计的黑盒测试用例的完整性，防止功能或功能组合的遗漏。
- 能够减小需求或设计不详细或不完整对测试造成的影响。

6.2 集成测试的实施

1. **集成测试的步骤**

为成功地实施集成测试，需遵循一定的步骤。集成测试的步骤如下。

(1) 计划集成测试

确定集成测试的内容和策略，安排集成测试的进度、资源、人员，确定测试中所需测试工具及硬件设备，对集成测试计划进行评审。

(2) 设计集成测试

创建集成测试环境，设计集成测试用例，细化测试过程。

(3) 执行集成测试

执行集成测试,并记录测试过程和发现的错误,定位并排除错误,必要时进行回归测试。

(4) 分析测试结果并提交测试报告

对集成测试的结果进行分析、归类,判断集成测试的完备性,编制并提交集成测试报告。

2. 集成测试计划的编制

集成测试与单元测试不同,单元测试是对单个模块的测试,被测单元的规模小,多个模块的单元测试可同时进行,无所谓顺序;但集成测试是一个对模块边集成边测试的过程,特别讲求对测试过程的精心计划。

在制定集成测试计划时,应主要考虑如下因素:

- 测试的内容;
- 集成测试中的系统集成方式;
- 集成测试过程中连接各模块的顺序;
- 模块代码编制和测试进度是否与集成测试的顺序一致;
- 测试过程中需要的测试工具及硬件设备。

3. 集成测试完成的标志

集成测试完成的标志如下:

(1) 成功地执行集成测试计划中规定的所有测试。

(2) 修正了集成测试中发现的错误。

(3) 测试结果通过了专门小组的评审。

6.3　集成测试的策略

集成测试的策略将直接影响集成测试用例的设计、测试工具的选择、测试的顺序和集成测试的成本等,在集成测试中具有极端重要性。

6.3.1　典型的集成测试策略

1. 一次性集成方式

一次性集成方式又称为大爆炸集成(Big-Bang Integration),是一种非增殖式集成方式(Non-Incremental Integration)。该小节中介绍的其他集成测试策略则属于增殖式集成方式(Incremental Integration)。

一次性集成方式的策略是,首先分别对每个模块进行单元测试,然后一次性地将所有模块集成在一起,并对它们进行测试,发现并清除在模块连接过程中出现的问题,得到最终要求的软件系统。

例如,有一个软件的模块结构如图 6-1(a)所示,其单元测试和集成顺序如图 6-1(b)所示。

在图 6-1(b)中,模块 d1,d2,d3,d4,d5 是对各模块实施单元测试时建立的驱动模块,

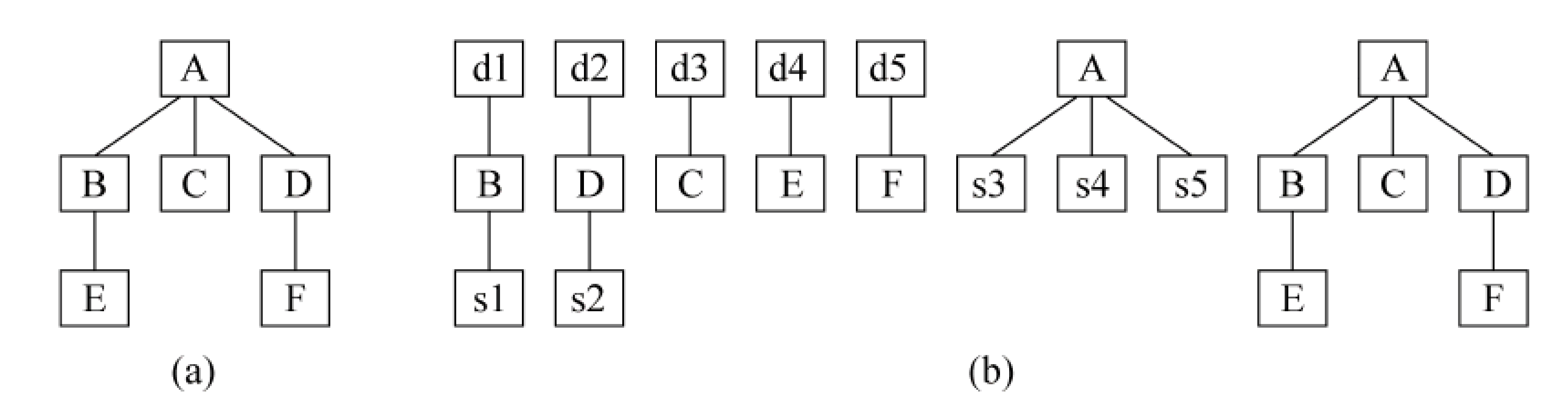


图 6-1　一次性集成的例子

s1,s2,s3,s4,s5 则是为各模块实施单元测试建立的桩模块。这种一次性集成方式试图在驱动模块和桩模块的辅助下,一次性地将通过了单元测试的各模块集成起来,并对模块的接口进行测试。

当软件的模块数量较多、接口复杂时,一次性集成方式不利于定位和解决发现的问题,所以很难构造成一个成功的最终系统。在实际应用中,该方式较少使用。

2. 自顶向下的集成方式

自顶向下的集成方式(Top-Down Integration)根据软件的模块结构图,按控制层次从高到低的顺序对模块进行集成,也就是从最顶层模块向下逐步集成,并在集成的过程中进行测试,直至组装成符合要求的最终软件系统。

自顶向下的集成方式的测试步骤如下:

① 以主模块为被测模块,主模块的直接下属模块则用桩模块代替。

② 采用深度优先或宽度优先策略,用实际模块替换相应的桩模块(注意每次仅替换一个或少量几个桩模块,视模块接口的复杂程度而定),它们的直接下属模块则又用桩模块代替,与已测试的模块或子系统集成为新的子系统。

③ 对新形成的子系统进行测试,发现和排除模块集成过程中引起的错误,并做回归测试。

④ 若所有的模块都已集成到系统中,则结束集成,否则转步骤②。

对于图 6-1(a)所示的模块结构图,图 6-2 给出了对其采用自顶向下增殖方式进行集成的过程。

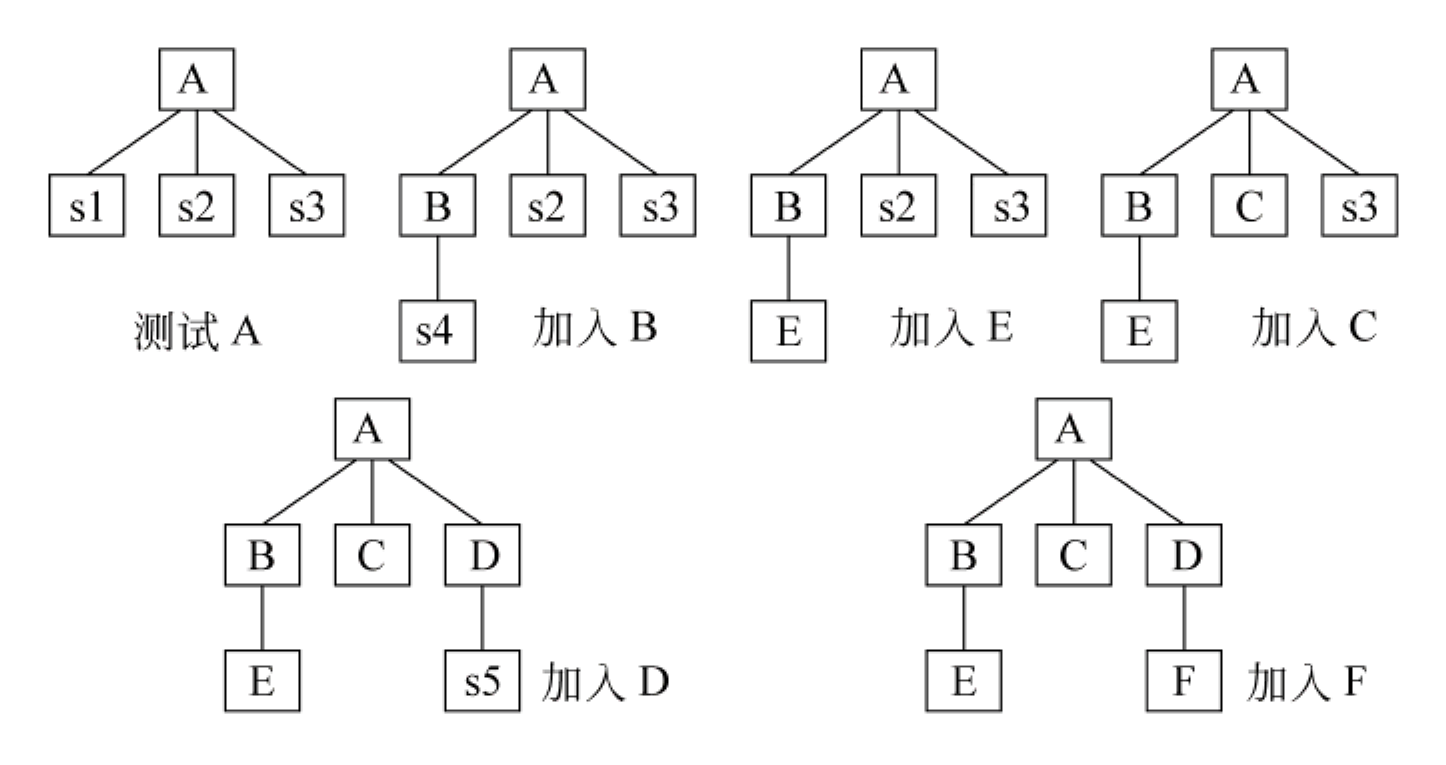


图 6-2　自顶向下集成的例子(按深度方向集成)

自顶向下的集成方式主要有如下优点：

- 可以及早地发现和修复模块结构图中的主要控制点存在的问题，以减少以后的返工，因为在一个模块划分合理的模块结构图中，主要的控制点多出现在较高的控制层次上。
- 能较早地验证功能的可行性。
- 最多只需一个驱动模块，减少了驱动模块的开发成本。
- 支持故障隔离。如某模块 A 通过了测试，而加进模块 B 后，测试中出现错误，则可以肯定错误处于模块 B 内部或模块 A、B 的接口上。

自顶向下集成方式的主要缺点如下：

- 需要开发和维护大量的桩模块。
- 桩模块很难模拟实际子模块的功能，而涉及复杂算法和真正输入/输出的模块一般在底层，它们是最容易出问题的模块，到组装的后期才测试这些模块，一旦发现问题，将导致大量的回归测试。
- 为了有效地进行集成测试，软件系统的控制结构应具有较高的可测试性。
- 随着测试的逐步推进，组装的系统愈加复杂，易导致对底层模块测试的不够充分，尤其是那些被复用的模块。

在实际使用中，自顶向下的集成方式很少单独使用，这是因为该方法需要开发大量的桩模块，增加了集成测试成本，违背了应尽量避免开发桩模块的原则。但在某些情况下，也可考虑使用该集成方式，例如：

- 软件的控制结构具有较大的技术风险，需较早得到测试；
- 在极限编程中使用探索式开发风格。注意，该方法要求测试人员在全部软件单元实现之前完成核心软件部件的集成测试。

3. 自底向上的集成方式

自底向上的集成方式(Down-Top Integration)根据软件的模块结构图，按控制层次从低到高的顺序对模块进行集成，也就是从最底层模块向上逐步集成，在集成的同时进行测试，直至组装成符合要求的最终软件系统。

因为是自底向上进行组装，对于一个给定层次的模块，它的所有下属模块已经组装并测试完成，所以不再需要桩模块。

自底向上集成方式的测试步骤如下：

① 为最底层模块(叶子模块)开发驱动模块，对最底层模块进行测试，最底层模块之间的测试可并行。也可将两个或多个最底层模块(模块数目视模块复杂程度而定)合并到一起作为被测模块，或将仅有一个叶子模块的父模块与叶子合并到一起做被测模块，为之开发驱动模块，进行测试。

② 用实际模块替换驱动模块，与其直属子模块(已被测试过)集成为一个子系统。

③ 为新形成的子系统开发驱动模块(若新形成的子系统对应为主控模块，则不必开发驱动模块)，对该子系统进行测试。

④ 若该子系统已对应为主控模块，即最高层模块，则结束集成，否则转步骤②。

对于图 6-1(a)所示的软件模块结构图，图 6-3 表示了对其进行自底向上集成的过程。

自底向上集成方式的主要优点如下：

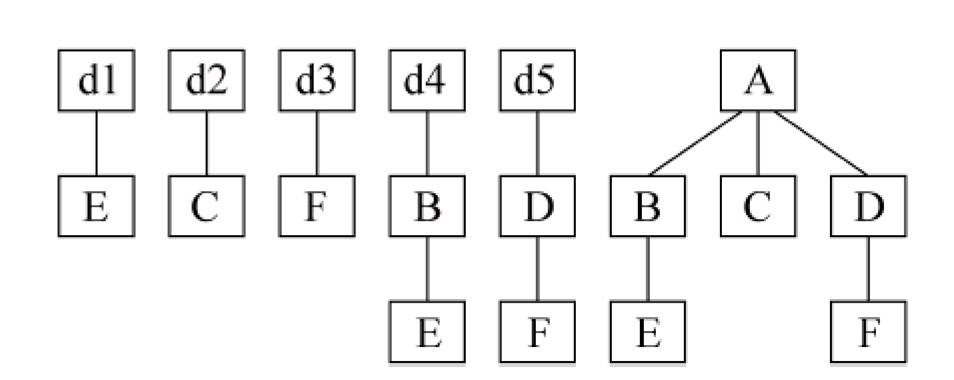


图 6-3 自底向上的集成方式

- 大大减少了桩模块的开发(但为了模拟一些中断和异常，可能仍需设计少量桩模块)，虽然需要开发大量的驱动模块，但其开发成本毕竟比开发桩模块的成本要小。
- 涉及复杂算法和真正输入/输出的模块往往在底层，它们是最容易出现问题的模块，最先对底层模块进行测试，减少了回归测试的成本。当然，若在集成测试后期又对底层模块进行修改，则必须对其上层模块进行回归测试，但这样的情况毕竟是少数。
- 在集成的早期很可能实现对模块的并行测试，这提高了集成测试的效率。
- 支持故障隔离。

自底向上集成方式的主要缺点如下：

- 需开发大量的驱动模块，故带来一定的测试成本。但通过对底层模块的复用可以减少驱动模块的开发。
- 不能及早地发现和修复模块结构中的主要控制点存在的问题，在集成的后期修复此类问题将花费较大的成本，故此法不适合于那些控制结构对整个体系至关重要的软件产品。
- 随着测试的逐步推进，组装的系统愈加复杂，对底层模块的异常很难测试到。

在实际的使用中，自底向上集成方式比自顶向下集成方式应用更广泛，尤其是在如下场合，更应使用自底向上的集成方式。

(1) 软件的高层接口变化较频繁，可测试性不强。

(2) 软件的底层接口较稳定。

4. 混合式集成方式

这种集成测试策略结合了自顶向下集成方式和自底向上集成方式的优点，在对一个软件的集成测试过程中，综合地使用此两种集成方案。例如首先对含有输入/输出模块和引入新算法模块等关键模块的子系统进行自底向上的组装和测试，然后由主模块开始自顶向下进行测试。又如首先对含读操作的子系统自底向上直至根结点模块进行组装和测试，然后对含写操作的子系统做自顶向下的组装与测试。

5. 核心系统先行集成方式

该方式的思想是先对核心软件部件进行集成测试，在此基础上再按各外围软件部件的重要程度逐个集成到核心系统中。每次加入一个外围软件部件都产生一个产品基线，直至最后形成稳定的软件产品。核心系统先行集成方式对应的集成过程是一个逐渐趋于闭合的螺旋形曲线，代表产品逐步定型的过程。

核心系统先行集成方式的步骤如下：

① 保证核心系统中的每个模块都已经通过单元测试。

② 对于核心系统中的所有模块一次性集合到被测系统中，解决集成中出现的各类问

题。在核心系统规模相对较大的情况下，可以采用自底向上的策略，集成核心系统的各模块。

③ 按照各外围软件部件的重要程度及模块间的相互制约关系，确定外围软件部件集成到核心系统中的顺序。

④ 在外围软件部件集成到核心系统之前，应首先对外围软件部件内部各模块完成集成测试。

⑤ 按顺序不断加入外围软件部件，排除外围软件部件集成中出现的问题，最终形成可正常运行的目标系统。

一般说来，核心系统先行集成方式是一种混合式集成方式，也就是既用到了自顶向下的集成方式，也用到了自底向上的集成方式，以充分利用两者的优点。

该集成方式对于快速软件开发很有效果，适合较复杂系统的集成测试，能保证一些重要功能的实现。

核心系统先行集成方式的缺点是被测软件一般必须具有如下特点方能使用此法：能明确区分核心软件部件和外围软件部件；核心软件部件内部各模块具有较紧密的关系；外围软件部件内部也具有较紧密的关系；而各外围软件部件之间具有较低的耦合度。

6. 高频集成方式

高频集成（High-frequency Integration）方式是指同步于软件开发过程，每隔一段时间对开发团队的现有代码进行一次集成测试。例如某些自动化集成测试工具能实现每日深夜对开发团队的现有代码进行一次集成测试，然后将测试结果发到各开发人员的电子信箱中。该集成测试方法频繁地将新代码加入到一个已经稳定的基线中，以发现集成中的故障，同时控制可能出现的基线偏差。

高频集成方式需具备一定的执行条件如下：

（1）可以持续获得一个稳定的增量，并且该增量内部已经验证没有问题。

（2）大部分有意义的功能增加可以在一个相对稳定的时间间隔（如每个工作日）内获得。

（3）测试包和代码的开发工作必须是并行进行的，并且需要版本控制工具来保证始终维护的是测试脚本和代码的最新版本。

（4）由于高频集成的集成次数频繁，必须借助于自动化工具来完成。

高频集成方式的步骤如下：

① 选择集成测试自动化工具。例如很多 Java 项目采用 JUnit 和 Ant 进行增量开发，实现集成测试的自动化。

② 设置版本控制工具，以确保集成测试自动化工具所获得的版本是最新版本。例如使用 CVS 进行版本控制。

③ 测试人员和开发人员负责编写对应程序代码的测试脚本。

④ 设置自动化集成测试工具，每隔一段时间对配置管理库的新添加的代码进行自动化的集成测试，并将测试报告汇报给开发人员和测试人员。

⑤ 测试人员监督代码开发人员及时关闭不合格项。

⑥ 重复步骤③～⑤，直至形成最终软件产品。

高频集成方式的优点主要如下：

- 能在开发过程中及时发现代码错误，尽早搭建一个可运行的系统，以直观地看到开发团队的进度。
- 测试中发现的错误很可能存在于新增或新近被修改的代码中，这对定位并修复错误很有益处。
- 开发维护源代码与开发维护软件测试包被赋予同等的重要性，这对有效防止和纠正错误都很有利。
- 对桩模块不是必须的，减少了测试成本。

该方案的缺点如下：

- 测试包有时候过于简单，可能不能暴露深层次的编码错误和图形界面错误。
- 在刚开始的几个周期可能不易于平稳地进行集成。

高频集成多用于快速迭代式开发或增量式开发中。

7. 基于消息/事件/线程的集成方式

对于许多基于状态机的系统，如嵌入式系统、面向对象方式开发的系统，模块（如类、进程）间的接口主要通过消息来实现，因而验证消息路径的正确性在这类软件系统的集成测试中具有重要的意义。集成测试可采用基于消息/事件/线程的集成方式（Message/Event/Thread-Based Integration）。

基于消息/事件/线程集成方式的步骤如下：

① 从软件系统的外部进行分析，确定可能输入的消息的集合。

② 选取一条消息，分析其穿越的所有模块。

③ 将这些模块进行集成，测试消息接口。

④ 选取下一条消息，重复步骤②、③，直至所有模块都被集成进来。

在消息集合中选取一条消息进行接口测试时，可从以下几方面考虑：

(1) 应对重要的消息路径优先进行测试。

(2) 选取的消息有利于新模块被集成进来。

(3) 选取路径较短的消息，以有效地对消息接口进行测试。

8. 基于使用的集成方式

在面向对象方式开发的软件系统中，有些类之间是独立的，而有些类之间具有依赖关系。基于使用的集成（Use-Based Integration）方式是，首先对各个类之间的依赖关系进行分析，测试独立的类（指几乎不使用服务器的类），然后测试使用一些服务器的类，再逐步测试具有依赖性的类（即使用独立类的类），直至整个系统构造完成，从而验证类之间接口的正确性。

6.3.2 集成测试策略的选取

以上介绍了一些常见的集成测试策略，一般来讲，一次性集成多方式用于系统规模较小的测试项目；自顶向下集成（较少被使用）、自底向上集成和混合式集成方式则多用于采用结构化方法开发的软件项目；基于消息/事件/线程的集成方式用于嵌入式系统、面向对象方式开发的系统，基于使用的集成方式用于面向对象系统；核心系统先行集成方式和高

频集成方式，则是广泛地被使用于许多复杂软件项目的集成测试过程。

在实际的测试项目中，使用的集成测试策略一般不是单一的，应结合软件项目的具体特点和开发环境合理地选择集成测试策略，以充分发挥各种方式的优点。

6.3.3　模块和接口的确定

模块的划分是在概要设计阶段就应该完成的工作，但在集成测试阶段，还应当从可测试性的角度进一步确定被测模块及接口。选定的被测模块和接口将直接影响集成测试的工作量、进度和效果。

为确定需集成的被测模块，可按以下思路进行：

① 确定当前主要希望测试的模块。

② 确定与该模块关系密切的模块，即有调用关系的模块，可按关系的紧密程度排列。

③ 将该模块与关系最紧密的模块进行集成。

④ 再依次考虑被集成模块的外围模块。

在集成测试中，较为合理的模块划分应具有如下特点：

- 被集成的模块关系紧密，能共同完成某些功能。
- 外围模块便于屏蔽，即被集成模块和外围模块的调用关系尽可能少，以减少驱动模块和桩模块的编写。
- 外围模块发给被集成模块的消息能模拟大部分情况。
- 模拟外围模块发给被集成模块的消息便于构造和修改。

集成测试是在一定的时间、资源和资金约束下进行的，不可能对软件中的模块和接口进行完全无遗漏的测试。应首先确定关键模块，尽早对关键模块及其接口进行测试。事实上，软件工程中的 2/8 原则在测试中是同样适用的，即测试中发现的 80%的错误可能位于软件的 20%区域内。也就是说，错误往往具有群集性，一个模块在测试中被发现的错误越多，往往意味着这个模块还可能有更多未被发现的错误。例如在 IBM OS/370 操作系统中，47%的软件故障(用户在使用中发现的)均出自 4%的软件模块。

因此，对关键模块的确定和尽早测试是提高集成测试效率的有效途径，集成测试活动应以关键模块为核心展开。

关键模块是具有以下特征之一的模块：

- 能够完成需求规格说明中的关键功能。
- 在软件模块结构中处于较高的层次。
- 较复杂，易出现错误。
- 有明确的性能要求。
- 被频繁使用(故易成为软件性能的瓶颈)。

对于非关键模块，可在后期对它们进行一次性集成测试。

6.4　小结

集成测试在软件测试中占有十分重要的地位，一般说来，集成测试花费的时间远远超过单元测试。集成测试往往是一个持续的过程，不是一蹴而就的，因而在执行集成测试前

合理地对其进行计划，尤其是根据软件项目的特点选择集成测试的策略，对于集成测试的成功实施有重要的意义。

本章介绍的主要是一些常规的集成测试策略。关于面向对象系统的集成测试是目前研究的热点，其中有很多种新方法或思路被提出。在本章对此涉及很少，将在第 8 章中详细介绍。

习 题

1. 简述集成测试的作用。
2. 为什么通过了单元测试的单元并不能保证在集成过程中不出现问题?
3. 集成测试的测试对象是什么?
4. 集成测试阶段的测试用例设计方法有哪些?
5. 为什么自顶向下的集成测试方式很少单独使用?
6. 为何在自底向上集成中进行回归测试的成本远小于自顶向下的集成?
7. 自底向上集成方式的优缺点分别是什么?
8. 混合式集成方式的目的是什么?
9. 高频集成方式的含义是什么? 一般适用于何种场合?
10. 面向对象开发一般使用何种集成测试策略?
11. 什么是关键模块? 为什么应尽早对关键模块及其接口进行测试?

第 7 章

确认测试、系统测试和验收测试

本章要点：

- 确认测试的任务。
- 确认测试活动。
- 系统测试的任务。
- 主要的系统测试类型。
- 性能测试的类型。
- 性能测试策略。
- 基于 Web 的系统测试。
- 回归测试的任务和策略。
- 系统测试的步骤。
- 验收测试的概念和策略。

经过集成测试后，应对软件产品进行最后阶段的测试——确认测试、系统测试和验收测试，并对测试中发现的缺陷进行必要的修复，方能将软件交付用户使用。在这一章，将介绍与确认测试、系统测试及验收测试相关的概念和策略。

7.1 确认测试的概念和活动

1. 确认测试的概念

经过集成测试后，软件产品已基本定型。确认测试(Validation Testing)的任务是验证软件的功能、性能及其他特性是否达到需求规格说明书的要求。在需求规格说明书中描述了软件中用户可见的所有属性，其中有一节为有效性准则，它就是软件确认测试的基础。在测试规格说明书(Test Specification)中，则会对需求规格说明中的要求做进一步的细化，用于指导确认测试的进行。确认测试也被软件开发企业称为合格性测试。

确认测试一般不由软件开发人员执行，而应由软件企业中独立的测试部门或第三方测试机构完成。

2. 确认测试活动

确认测试一般包括有效性测试和软件配置复查。以下分别介绍之。

(1) 有效性测试

有效性测试是在模拟的环境下,通过执行黑盒测试,验证被测软件是否满足需求规格说明书中的需求。需求规格说明书中的需求是多方面的,有对功能的需求,对性能的要求,对文档的需求,以及对其他特性(如安全性、健壮性、兼容性等)的要求。

为进行有效性测试,应拟订测试计划,详细指明要执行的测试种类、测试的步骤,还需要编写具体的测试用例。

在执行完所有测试用例后,应对本次测试做出结论。结论分为以下两种:

① 测试结果与预期结果相符,即软件的功能、性能及其他特性满足需求规格说明书中的需求,即通过了有效性测试。

② 测试结果与预期结果不相符,即软件的功能、性能及其他特性未能满足需求规格说明书中的需求,应提交一份问题报告。

(2) 软件配置复查

在软件工程过程中产生的所有信息项,如文档、报告、程序、表格、数据等,称为软件配置。软件配置复查的目的是保证软件配置的所有成分齐全,各成分的质量都符合要求,具有维护阶段所必需的细节,且已编排好分类目录。

在确认测试的过程中,应严格按照用户手册和操作手册中规定的使用步骤,检查软件配置是否齐全、正确。应详细记录发现的遗漏或不正确之处,并对发现的问题进行修复。

7.2 系统测试的概念和类型

7.2.1 系统测试的概念

1. 系统测试的含义

最终得到的软件不是一个孤立的对象,往往是作为整个目标系统(注意前文所说的系统是指软件系统,而此处却不是)的一个组成元素而存在。系统测试(System Testing)是将已经通过确认测试的软件作为基于整个计算机系统的一个元素,与系统中其他所有元素(包括硬件、外设、网络、系统软件、支持平台、数据、操作人员等)结合在一起,在实际运行环境下或模拟系统运行环境下,测试其与系统中其他元素能否实现正确地连接,以满足用户的所有需求。

系统测试的任务是通过与系统的需求定义比较,发现软件与系统的定义不符合的地方。

显然,系统测试已超出了对软件进行测试的范围,但软件在整个系统中往往占据重要的位置,软件的质量对系统测试的成功与否有极大的影响。一个软件能够顺利地通过系统测试,是对其质量的最佳诠释。

与确认测试一样,系统测试一般不由软件开发人员执行,而应由软件企业中独立的测试部门或第三方测试机构完成。

系统测试阶段使用黑盒方法设计测试用例，完成对整个系统的测试。

2. 系统测试与确认测试的关系

在很多资料中，确认测试被认为是系统测试的一部分，业界也常将确认测试并入系统测试中进行讨论。从广义上说，这样理解是可以的，但狭义的系统测试却与确认测试有着较大的不同，这种不同表现为：确认测试一般以功能测试为主，而在系统测试中，一般以性能测试为主，如压力测试、负载测试、强度测试、容量测试等都属于性能测试。

7.2.2　系统测试的主要类型

系统测试在整个测试过程中处于收尾阶段，其种类繁多，除了功能测试外，主要包括如下测试类型。

1. 性能测试

性能测试的目的是检测系统在一定的工作环境中，能否达到预期的性能要求。性能测试主要关注被测系统的事务处理速率和响应时间等性能指标。例如，压力测试和负载测试、疲劳测试、强度测试、容量测试等都是常见的性能测试类型。下面分别进行介绍。

(1) 压力测试和负载测试

压力测试(Stress Testing)是改变应用程序的输入，以对应用程序施加越来越大的负载，通过综合分析交易执行指标和资源监控指标，评测和评估应用系统在不同负载条件下的性能的行为。

压力测试的目的主要体现在以下三个方面：

- 以真实的业务为依据，选择有代表性的、关键的业务操作设计测试案例，以评价系统的当前性能。
- 当扩展应用程序的功能或者新的应用程序将要被部署时，压力测试帮助确定系统是否还能够处理期望的用户负载，以预测系统的未来性能。
- 通过模拟成百上千个用户，重复执行和运行测试，可以确认性能瓶颈，获得系统能提供的最大服务级别，并调整应用以优化其性能。例如，测试一个 Web 站点在大量的负荷下，何时系统的响应会退化或失败。

关于负载测试(Load Testing)和压力测试的区别，人们的说法不尽相同，其实也没有必要去深究两者的区别，这不会影响实际的测试效果，只要我们能针对具体的应用系统和用户需求制定合理、全面、详尽的压力(或负载测试)方案。

压力测试和负载测试也可理解为并发性能测试，这类测试最常见于采用联机事务处理(OLTP)方式的数据库应用、Web 浏览和视频点播等系统，是性能测试中的重点。

下面来看一个电信计费软件的例子。

每个月市话交费的高峰期，全市几千个收费网点同时启动。收费过程一般分为两步，首先要根据用户给出的电话号码查询出其当月产生费用，然后收取现金并将此用户修改为已交费状态。一个用户看起来简单的两个步骤，但当成百上千的终端同时执行这样的操作时，情况就大不一样了，如此众多的交易同时发生，对应用程序本身、操作系统、中心数据库服务器、中间件服务器、网络设备的承受力都是一个严峻的考验。若该电信计费软件在使用过程中不能达到应有的性能或是因负载过大而发生性能急剧下滑，则会给缴费

者和操作人员带来麻烦，同时给电信部门带来收益的损失。因而必须在该应用系统投放前就进行严格的负载测试。预先知道了应用系统能承受的最大负荷，就为最终用户规划整个运行环境的配置提供了有力的依据，以避免不必要的损失。

用人工方法进行负载测试是不切实际的，且无法捕捉到应用程序内部的变化情况。负载测试主要是通过测试工具完成的，也就是进行自动负载测试，即通过在一台或几台PC机上模拟成百或上千个虚拟用户同时执行业务的情形，对应用系统进行测试，同时记录下每一事务处理的时间、中间件服务器峰值数据、数据库状态等。通过执行可重复的、尽量接近实际使用环境的测试，测量和分析应用系统的可扩展性和性能，确定性能瓶颈，以优化系统性能。

执行负载测试的准备工作主要如下。

① 建立测试环境。建立测试环境是测试实施的必要前提，测试环境的适合与否会严重影响测试结果的真实性和正确性。

测试环境包括硬件环境和软件环境，硬件环境指测试所必须的服务器、客户端、网络连接设备，以及打印机/扫描仪等辅助硬件设备所构成的环境；软件环境指被测软件运行时的操作系统、数据库及其他应用软件构成的环境。

一个充分准备好的测试环境应具备以下三个特点：

- 稳定、可重复的测试环境，能够保证测试结果的准确；
- 保证达到测试执行的技术需求；
- 能保证测试结果的易理解性。

② 选择或开发测试工具。负载测试是在客户端执行的黑盒测试。目前有很多测试工具可用于负载测试，包括商业性的和免费的，当然也可以组织人员自行开发。关于自动化负载测试工具，将在第9章介绍。

负载测试工具可以在整个开发生命周期，跨越多种平台、自动地执行测试任务，模拟成百上千用户并发执行关键业务，从而完成对应用系统的测试。选择负载测试工具的依据主要是测试需求和性能价格比。

③ 准备测试数据。在初始的测试环境中需要输入一些适当的测试数据，目的是识别数据状态并验证用于测试的测试案例，在正式的测试开始以前对测试案例进行调试，将正式测试开始时的错误降到最低。在测试进行到关键过程环节时，非常有必要进行数据状态的备份。制造初始数据意味着将合适的数据存储下来，在需要的时候恢复它，初始数据提供了一个基线（基线配置又称基准配置，它们是经过阶段评审后的软件配置成分）用来评估测试执行的结果。

在测试正式执行时，还需要准备业务测试数据，如测试并发查询业务就要求对应的数据库和表中有相当的数据量及数据的种类应能覆盖全部业务。

采用自动化负载测试工具执行的并发性能测试，基本测试过程如下：

① 确定测试需求、测试内容。

② 制定测试案例。

③ 配置测试环境

④ 测试脚本录制、编写与调试，制定脚本分配、回放配置与加载策略。

⑤ 执行并跟踪测试。

⑥ 分析测试结果,定位问题,完成测试报告和评估。

负载测试(或并发性能测试)监控的对象不同,测试的主要指标也不相同,主要的测试指标包括交易处理性能指标和 UNIX 资源监控。其中,交易处理性能指标包括交易结果、每分钟交易数、交易响应时间(Min:最小服务器响应时间;Mean:平均服务器响应时间;Max:最大服务器响应时间;StdDev:事务处理服务器响应的偏差;Median:中值响应时间;90%:90%事务处理的服务器响应时间)和虚拟并发用户数。

(2) 疲劳测试

疲劳测试(或称疲劳强度测试)是采用系统稳定运行情况下能够支持的最大并发用户数。持续执行一段时间业务,通过综合分析交易执行指标和资源监控指标可以确定系统处理最大业务量时的性能。疲劳测试的主要目的是测试系统的稳定性,同时它也是对应用系统并发性能的测试。

介绍了压力(负载)测试和疲劳测试的概念后,来看一个某通信社多媒体数据库 V 1.0 并发性能测试的实例。

某软件评测中心根据某通信社技术局提出的《多媒体数据库(一期)性能测试需求》和 GB/T 17544《软件包质量要求和测试》国家标准,使用工业标准级负载测试工具对该社使用的多媒体数据库 V1.0 进行了性能测试。

① 性能测试的目的。模拟多用户并发访问该多媒体数据库,执行关键检索业务,分析系统性能。

② 性能测试的重点。针对系统并发压力较大的主要检索业务,进行负载测试和疲劳测试,系统采用 B/S 运行模式。负载测试设计了特定时间段内分别在中文库、英文库、图片库中进行单检索词、多检索词,以及变检索式、混合检索业务等并发测试案例。疲劳测试案例为在中文库中并发用户数 200,进行测试周期约 8 小时的单检索词检索。在进行负载测试和疲劳测试的同时,监测的测试指标包括交易处理性能及 UNIX(Linux),Oracle,Apache 资源等。

③ 测试结果。在该通信社机房测试环境和内网测试环境中,100Mbps 带宽情况下,针对规定的各并发性能测试案例,系统能够承受并发用户数为 200 的负载压力,每分钟最大交易数达到 78.73,运行基本稳定。但随着负载压力增大,系统性能有所衰减。

系统能够承受 200 并发用户数持续周期约 8 小时的疲劳压力,基本能够稳定运行。

通过对系统 UNIX(Linux),Oracle 和 Apache 资源的监控,系统资源能够满足上述负载压力和疲劳性能需求,且系统硬件资源尚有较大利用余地。

当并发用户数超过 200 时,监控到 HTTP 500、连接和超时错误,且 Web 服务器报内存溢出错误。

④ 建议。进一步优化软件系统,充分利用硬件资源,缩短交易响应时间,支持更大并发用户数。

(3) 强度测试

强度测试(Intensity Testing)的目的是找出因资源不足或资源争用而导致的错误。如果内存或磁盘空间不足,测试对象就可能会表现出一些在正常条件下并不明显的缺陷;

而其他缺陷则可能是由于争用共享资源(如数据库锁或网络带宽)造成的。强度测试还可用于确定测试对象能够处理的最大工作量。

强度测试和压力测试的测试指标相近,大多都是与时间相关的指标,如响应时间、事务处理速率等。

(4) 容量测试

容量测试(Volume Testing)通常与数据库有关,其目的在于使系统承受超额的数据容量来确定系统的容量瓶颈(如同时在线的最大用户数),进而优化系统的容量处理能力。

容量测试的步骤如下:

① 分析系统的外部数据源,并进行分类。

② 分析系统对每类数据源的容量限制。

③ 根据分析结果,为每类数据源构造相应的大容量数据对系统进行测试。

④ 分析测试结果,确定系统的容量瓶颈。

⑤ 对系统进行优化,并重复执行步骤①~④,直至达到预期的容量处理能力。

比如,在电子商务系统中,通过往 user 表中插入 10000 数据,看其是否可以正常显示顾客信息列表页面。如果要求达到最多可以处理 100000 个客户,但是顾客信息列表页面不能够在规定的时间内显示出来,就需要调整程序中的 SQL 查询语句;如果在规定的时间内显示出来,可以将用户数分别提高到 20000, 50000, 100000 进行测试,以找到系统的容量瓶颈。

2. 安全性测试

安全性测试(Security Testing)的目的在于检测软件系统对非法侵入的防范能力。它是通过模拟软件真实运行环境下攻击者的操作行为,如通过力图截取或破译系统的口令、破坏系统的保护机制、导致系统出现故障并在系统恢复过程中企图非法进入、通过浏览非保密数据以试图推导所需的保密信息等来寻找软件架构中不合理之处和编码的安全隐患。安全性测试包括验证输入数据、破解加密和访问敏感数据、缓冲区溢出、授权和证书功能等。

从理论上说,只要拥有足够的时间和资源,任何系统都是可以进入的。所以,通过安全性测试及采取相应的措施后,应使非法侵入软件系统的代价大于被保护信息的价值,使非法侵入者得不偿失。

3. 健壮性测试

健壮性是指在异常情况下,软件能继续正常运行的能力。健壮性有两层含义:一是容错能力,二是恢复能力。因而健壮性测试(Robustness Testing)包括容错性测试和恢复性测试。

(1) 容错性测试

进行容错性测试(Fault Tolerance Testing)时,通常构造一些不合理的输入来引诱软件出错,观察其能否继续正常工作。例如,输入不合理的月份,输入与数据类型要求不符的数据;又如在测试 C/S 模式的软件时,把网络线拔掉造成通信异常中断等。

(2) 恢复性测试

恢复性测试(Recoverability Testing)的含义是将系统置于极端条件下(或者是模拟的极端条件下),迫使其发生故障(如设备 I/O 故障或无效的数据库指针和关键字),检测系统恢复正常工作状态的能力。

若系统能自动进行恢复,应检查的项目包括:重新初始化、检验点设置机构、数据恢复及重新启动;若需人工干预进行恢复,还需测试系统的平均修复时间,判断其是否在限定的时间范围内。

恢复性测试还需对系统的故障转移能力进行评判。故障转移指当主机软硬件发生故障时,备份机器能及时启动,使系统继续正常运行,以避免丢失任何数据或事务。这对于电信、银行等领域处理重要事务的软件是十分重要的。

4. 可靠性测试

可靠性(Reliability)是指在一定的环境下,系统不发生故障的概率。由于软件不像硬件那样可以加速老化,软件可靠性测试(Reliability Testing)可能会花费很长时间。

为解决这一问题,比较实用的方法是让用户使用系统,记录每一次发生故障的时刻。计算出相邻故障的时间间隔(注意去除非工作时间)。这样便可以方便地统计出不发生故障的最小时间间隔、最大时间间隔和平均时间间隔。其中,平均时间间隔也可称为平均无故障时间,在很大程度上代表了软件系统的可靠性。

5. 配置测试和兼容性测试

兼容性测试(Compatibility Testing)有时也被称为配置测试(Configuration Testing),但它们的含义略有不同。一般说来,配置测试的目的是保证软件在其相关的硬件上能够正常运行,而兼容性测试主要是测试软件能否与其他软件协作运行。

配置和兼容性测试通常对开发系统类软件比较重要,如驱动程序、操作系统、数据库管理系统等。

配置测试的核心内容就是使用各种硬件来测试软件的运行情况,一般包括如下:

- 软件在安装不同类型 CPU 的机器上的运行情况。
- 软件在安装不同厂商的浏览器时的运行情况。
- 软件在不同组件上的运行情况,例如开发的拨号程序要测试在不同厂商生产的 Modem 上的运行情况。
- 不同的外设。
- 不同的接口。
- 不同的可选项,如不同的内存容量等。

兼容性测试的核心内容如下:

- 软件是否能在不同的操作系统平台上兼容。
- 软件是否能在同一操作系统平台的不同版本上兼容。
- 软件本身能否向前或者向后兼容。
- 软件能否与其他相关的软件兼容。
- 数据兼容性测试,即测试能否与其他软件共享数据。

6. 用户界面测试

目前，大多数软件都具有图形用户界面GUI(Graphic User Interface)。因为GUI开发环境有可复用的构件，开发用户界面更加省时而且更加精确，但GUI的复杂性也增加了，从而加大了设计和执行测试用例的难度。GUI测试的重点是图形用户界面的正确性、易用性和视觉效果。

因为现在GUI设计和实现有了越来越多的类似，所以也就产生了一系列测试标准。下列问题可以作为常见GUI测试的指南。

(1) 窗口

① 窗口是否能够基于相关的输入和菜单命令适当地打开。

② 窗口能否改变大小、移动和滚动。

③ 窗口中的数据内容能否用鼠标、功能键、方向键和键盘访问。

④ 当被覆盖并重新调用后，窗口能否正确地重现。

⑤ 需要时能否使用所有窗口相关的功能。

⑥ 所有窗口相关的功能是否可操作。

⑦ 是否有相关的下拉式菜单、工具条、滚动条、对话框、按钮、图标和其他控件可为窗口使用，并适当地显示。

⑧ 显示多个窗口时，窗口的名称是否能够被适当地表示。

⑨ 活动窗口是否能够被适当地加亮。

⑩ 如果使用多任务，是否所有的窗口被实时更新。

⑪ 多次或不正确按鼠标键是否会导致无法预料的副作用。

⑫ 窗口的声音和颜色提示及窗口的操作顺序是否符合需求。

⑬ 窗口能否被正确地关闭。

(2) 下拉式菜单和鼠标操作

① 菜单条是否显示在合适的上下文环境中。

② 应用程序的菜单条是否显示与系统相关的特性(如时钟显示)。

③ 下拉式操作能否正确工作。

④ 菜单、调色板和工具条是否能够正确工作。

⑤ 是否适当地列出了所有的菜单功能和下拉式子菜单功能。

⑥ 是否可以通过鼠标访问所有的菜单功能。

⑦ 文本字体、大小和格式是否正确。

⑧ 是否能够用其他文本命令激活每个菜单功能。

⑨ 菜单功能是否随当前的窗口操作加亮或变灰。

⑩ 菜单功能能否正确执行。

⑪ 菜单功能的名字是否具有自解释性。

⑫ 菜单项是否有帮助，是否上下文环境相关。

⑬ 在整个交互式环境中，是否可以识别鼠标操作。

⑭ 如果要求多次点击鼠标，是否能够在上下文环境中正确识别。

⑮ 光标、处理指示器和识别指针是否能够随操作恰当地改变。

(3) 数据项

① 字母数字数据项是否能够正确回显,并输入到系统中。

② 图形模式的数据项(如滚动条)是否能够正常工作。

③ 是否能够识别非法数据。

④ 数据输入消息是否可理解。

7. 文档测试

文档测试(Documentation Testing)主要是指对提交给用户的文档进行的测试。这是一项十分重要的测试。文档测试的对象主要包括:包装文字和图形,市场宣传材料、广告及其他插页,授权、注册登记表,最终用户许可协议,安装和设置向导,用户手册,联机帮助,样例、示例和模板。

文档测试的目的是提高易用性和可靠性,降低技术支持费用,尽量使用户通过文档自行解决问题。因此,文档测试的检查内容主要包括如下:

(1) 文档的内容是否能让不同级别的读者理解。

(2) 文档中的术语是否适合读者。

(3) 内容和主题是否合适。

(4) 图表的准确度和精确度如何。

(5) 样例和示例是否与软件功能一致。

(6) 拼写和语法是否准确。

(7) 文档是否与其他相关文档的内容一致,例如与广告信息是否一致。

8. 安装测试和卸载测试

(1) 安装测试

安装测试(Installation Testing)的目的是确认如下方面能否实现:

① 安装程序能够正确运行。

② 软件安装正确。

③ 软件安装后能够正常运行。

安装测试应着重关注如下方面:

① 安装手册中的所有步骤得到验证。

② 安装过程中所有默认选项得到验证。

③ 安装过程中典型选项得到验证。

④ 测试各种不同的安装组合,并验证各种不同组合的正确性,包括参数组合、控件执行顺序组合、产品安装组件组合、产品组件安装顺序组合(如 B/S)等。

⑤ 对安装过程中异常配置或状态(非法和不合理配置)情况进行测试,例如断电、网络失效、数据库失效等。

⑥ 安装后是否能生成正确的目录结构和文件。

⑦ 安装后动态库是否正确。

⑧ 安装后软件能否正常运行。

⑨ 安装后是否会生成多余的目录结构、文件、注册表信息、快捷方式等。

⑩ 安装测试是否在所有的运行环境上进行了验证，例如操作系统、数据库、硬件环境、网络环境等。

⑪ 能否在笔记本上进行安装（很多产品在笔记本中安装时会出现问题，尤其是系统级产品）。

⑫ 安装软件后是否会对操作系统或某些应用程序造成不良影响。

⑬ 是否可以识别大部分硬件。

⑭ 确认打包程序的特性，不同的打包发布程序所支持的系统都是不一样的。

⑮ 空间不足（如安装过程中向安装盘放入大量文件）时安装情况如何。

（2）卸载测试

卸载测试（Uninstallation Testing）应重点关注如下方面：

① 在不同的卸载方式下卸载。例如程序自带卸载程序、系统的控件面板卸载、通过其他自动卸载工具（如优化大师）。

② 软件在运行、暂停、终止等各种状态时的卸载。

③ 非正常卸载情况，如在卸载软件过程中取消卸载进程，然后观察软件能否继续正常使用。

④ 冲击卸载。即在卸载的过程中中断电源，启动计算机后，重新卸载软件。

⑤ 在不同的运行环境（如操作系统、数据库、硬件环境、网络环境等）下进行卸载。

⑥ 能否在笔记本上进行卸载。

⑦ 卸载后是否对操作系统或其他应用程序造成不良影响。

⑧ 卸载过程中是否删除了系统应该保留的用户数据。

⑨ 卸载后系统能否恢复到软件安装前的状态，包括快捷方式、目录结构、动态链接库、注册表、系统配置文件、驱动程序、关联情况等。

7.2.3 系统测试与集成测试的区别

从对集成后的功能进行测试这一点来看，集成测试与其后进行的系统测试有一些相似之处，但两者有着明显的区别。

系统测试是对全部模块集成完毕的软件进行功能、性能及其他特性（如安全性、兼容性、健壮性等）的测试，检测其与系统中其他元素（硬件、外设、网络、系统软件、支持平台等）能否协同工作，以满足用户的各种需求。系统测试的主要依据是软件的需求规格说明书和相关的行业标准。

而在集成的过程中，对功能和性能的测试则是集成测试，集成测试的主要依据是软件的概要设计说明书。

相对于系统测试而言，集成测试具有如下优点：

- 集成测试的用例设计是从软件架构出发的，针对性较强，较易于发现错误并找出错误的原因和位置；
- 集成测试是在模块组装过程中进行的测试，能有效地模拟几乎所有的实际执行流程，故能更有效地发现软件中的错误；
- 解决在集成测试阶段发现的问题的成本要远小于系统测试阶段。

正是由于集成测试相对于单元测试和系统测试有着自身的独特优势，使得集成测试

成为软件测试中一个不可或缺的环节。

7.3　性能测试策略

通过刚才对性能测试中的几种主要测试类型(压力测试、负载测试、疲劳测试、强度测试、容量测试)的介绍可以发现,不同的性能测试类型所做的工作却有很大的关联。事实上,性能测试的很多内容可以经过一定的组织统一进行,也就是可以按照"全面性能测试模型"的策略来开展性能测试,这样做的好处在于可以由浅入深地按照层次对应用系统进行测试,进而减少不必要的工作量,以实现节约测试成本。

全面性能测试模型提出的主要依据就是一种类型的性能测试可以在某些条件下转化成另外一种类型的性能测试,而这些类型的测试实施是非常类似的。例如,针对一个网站进行测试,模拟 10～50 个用户时就是在进行常规性能测试;而用户增加到 1000 乃至上万就变成了压力/负载测试;如果同时对系统进行大量的数据查询操作,就包含了强度测试或容量测试;若负载测试持续较长时间,则包含了疲劳测试。

7.3.1　全面性能测试模型

如今的性能测试,多数是针对 Web 系统的性能测试。在全面性能测试模型中,针对 Web 性能测试,将其划分为预期指标的性能测试、独立业务性能测试、组合业务性能测试、疲劳测试、容量测试、网络性能测试、服务器性能测试和特殊测试 8 个类别。当然,这并不影响全面性能测试模型在其他类型软件的性能测试中的运用。

下面首先介绍 8 个性能测试类别的主要内容。

1. 预期指标的性能测试

系统在需求分析和设计阶段都会提出一些性能指标。例如,系统可以支持并发用户 1000,系统响应时间不得高于 10 秒等在产品说明书等文档中十分明确的内容说明。

验证这些指标是性能测试要完成的首要工作之一,可将针对预先确定的性能指标的测试称为预期指标的性能测试。

2. 独立业务性能测试

独立业务主要是指一些核心业务模块,这些模块通常具有功能比较复杂、使用比较频繁、属于核心业务等特点。这类特殊的、功能比较独立的业务模块始终都是性能测试的重点。通常,不但要测试这类模块的一些与性能相关的算法,还要测试这类模块对并发用户的响应情况。

核心业务模块在需求阶段就可以确定,在系统测试阶段开始单独测试其性能。如果是系统类软件或者特殊应用的软件,通常从单元测试阶段就开始进行测试,在后继的集成测试、系统测试、验收测试中进一步进行测试,以保证核心业务模块的性能稳定。

并发性能测试是核心业务模块的重点测试内容,即模拟一定数量的用户同时使用某一核心模块的"相同"的或者"不同"的功能,并且持续一段时间。对"相同"的功能进行并发测试分为两种类型,一类是在同一时刻进行完全相同的操作,例如打开同一条数据记录进行查看;另外一类是在同一时刻使用完全相同的功能,例如同时提交数据进行保存。显

然，后者是包含前者的，而前者是后者的特例。

3. 组合业务性能测试

通常不会所有的用户只使用一个或者几个核心业务模块，因而 Web 性能测试既要模拟多用户的"相同"操作（即很多用户使用同一功能），又要模拟多用户的"不同"操作（即很多用户同时对一个或者多个模块的不同功能进行操作），对多个业务进行组合性能测试。组合业务测试是最接近用户实际使用情况的测试，因而是性能测试的核心内容。通常，按照用户的实际使用情况来模拟使用各个模板的人数比例。

由于组合业务测试是最能反映用户使用系统情况的测试，因而组合测试往往与服务器（操作系统、Web 服务器、数据库服务器）性能测试结合起来，在通过工具模拟用户行为的同时，还通过测试工具的监控功能采集服务器的计数器信息，进而全面分析系统的性能瓶颈，为优化系统性能提供有利的依据。

并发性能测试是组合业务测试的核心内容。"组合"并发的突出特点是分成不同的用户组进行并发，每组的用户比例要根据实际情况进行匹配。组合业务测试可以理解为包含"核心业务模块并发"和"非核心业务模块并发"同时进行的并发性能测试。

4. 疲劳测试

前文已经介绍过，疲劳测试是在系统稳定运行下模拟较大的用户数量，并长时间运行系统，通过综合分析执行指标和资源监控指标确定系统处理最大业务量时的性能。疲劳测试也属于并发性能测试，因而也可以分为独立业务的疲劳测试和组合业务的疲劳测试。

5. 容量测试

容量测试的目的在于使系统承受超额的数据容量确定系统的容量瓶颈。容量测试可以独立进行，也可以与前面介绍的独立、组合业务测试结合起来进行，主要由性能测试策略决定。

6. 网络性能测试

网络性能测试的重点是利用自动化测试工具进行网络应用性能监控、网络应用性能分析和网络预测。

（1）网络应用性能监控

在系统试运行之后，需要及时准确地了解网络上正在发生的事情；有哪些应用正在运行，如何运行；有多少 PC 正在访问 LAN 或 WAN；哪些应用程序会导致系统瓶颈或资源竞争，这时网络应用性能监控及网络资源管理对系统的正常稳定运行是非常关键的。利用网络应用性能监控工具，可以达到事半功倍的效果。这方面的工具有 Network Vantage 等。通俗地讲，它主要用来分析关键应用程序的性能，定位问题的根源是在客户端、服务器、应用程序还是网络。在大多数情况下，用户较关心的问题还有哪些应用程序占用大量带宽，哪些用户产生了最大的网络流量，这个工具同样能满足要求。

（2）网络应用性能分析

网络应用性能分析的目的是准确地展示网络带宽、延迟、负载和 TCP 端口的变化是如何影响用户的响应时间的。利用网络应用性能分析工具，例如 Application Expert，

能够发现应用的瓶颈，可知在网络上运行时在每个阶段发生的应用行为，在应用线程级分析应用的问题；可以解决多种问题，例如客户端是否对数据库服务器运行了不必要的请求，当服务器从客户端接受了一个查询，应用服务器是否花费了不可接受的时间联系数据库服务器；在投产前预测应用的响应时间；利用 Application Expert 调整应用在广域网上的性能；Application Expert 能够使用户快速、容易地仿真应用性能，根据最终用户在不同网络配置环境下的响应时间，用户可以根据自己的条件决定应用投产的网络环境。

（3）网络预测

考虑到系统未来发展的扩展性，预测网络流量的变化、网络结构的变化对用户系统的影响非常重要。从网络管理软件获取网络拓扑结构，从现有的流量监控软件获取流量信息（若没有这类软件可人工生成流量数据），可以得到现有网络的基本结构。在基本结构的基础上，可根据网络结构及网络流量的变化生成报告和图表，说明这些变化是如何影响网络性能的，对网络性能进行预测。

利用网络预测分析容量规划工具 Predictor 可以进行网络性能预测。Predictor 可根据预测结果帮助用户及时升级网络，避免因关键设备超过利用阈值导致系统性能下降；还可根据预测的结果避免不必要的网络升级。

7. 服务器性能测试

对服务器（操作系统、Web 服务器、数据库服务器）的性能测试可以采用工具监控，也可以使用系统本身的监控命令。例如，Tuxedo 中可以使用 Top 命令监控资源使用情况。

对于 UNIX 服务器，其主要的资源监控指标如表 7-1 所示。

表 7-1　UNIX 服务器的主要资源监控指标

监控指标	描　　述
平均负载	系统正常状态下，最后 60 秒同步进程的平均个数
冲突率	在以太网上监测到的每秒冲突数
进程/线程交换率	进程与线程之间每秒交换次数
CPU 利用率	CPU 占用率（%）
磁盘交换率	磁盘交换速率
接收包错误率	接收以太网数据包时每秒错误数
包输入率	每秒输入的以太网数据包数目
中断速率	CPU 每秒处理的中断数
输出包错误率	发送以太网数据包时每秒错误数
包输入率	每秒输出的以太网数据包数目
读入内存页速率	物理内存中每秒读入内存页的数目
写出内存页速率	每秒从物理内存中写到页文件中的内存页数目，或者从物理内存中删掉的内存页数目

续表

监控指标	描　　述
内存页交换速率	每秒写入内存页和从物理内存中读出页的个数
进程入交换率	交换区输入的进程数目
进程出交换率	交换区输出的进程数目
系统 CPU 利用率	系统的 CPU 占用率(%)
用户 CPU 利用率	用户模式下的 CPU 占用率(%)
磁盘阻塞	磁盘每秒阻塞的字节数

8. 特殊测试

主要是指配置测试、内存泄漏测试等特殊的 Web 性能测试。这类性能测试或者与前面讲述的测试结合起来使用,或者在一些特殊情况下独立使用(这时往往需要特殊的工具和较大的投入)。

7.3.2 性能测试策略

1. 性能测试策略

性能测试策略一般从需求分析及设计阶段开始制定,主要关注性能测试的内容、开始实施的时机、测试环境和工具的准备等。

性能测试策略制定的主要依据是应用系统的自身特点和用户对性能的关注程度,其中应用系统的自身特点又起着决定性作用。

软件可分为系统软件和应用软件。系统软件对性能一般要求比较高,因此性能测试应该尽早介入。一般可在需求分析及设计阶段就针对系统架构、数据库设计等方面进行讨论、规划,从根源提高应用的性能;并尽早制定性能测试策略,在单元测试阶段就应根据具体模块的性能需求开始实施相应的性能测试。

应用软件分为特殊类应用软件和一般类应用软件。特殊类应用主要指应用于金融、电信、电力、医疗、安全等领域的软件,这类软件使用比较频繁,且一旦发生性能问题,造成的损失较大,因而对性能测试要求一般也较高。对特殊类应用软件可以采取与系统软件类似的性能测试策略。

一般类应用主要指一些普通应用,例如办公自动化软件、MIS 系统等。一般应用类软件的业务重要级别较低,因此对性能测试的要求一般低于系统软件和特殊类应用软件,多根据实际情况决定性能测试策略。这类软件受用户主观因素即用户对性能测试的重视程度影响较大。如果用户对应用系统性能高度重视或比较重视,可采取与系统软件类似的性能测试策略;如果用户对应用性能不十分重视,则可在系统测试阶段的功能测试完成后再进行性能测试,或是在系统提交前进行性能测试。

一般类应用软件对性能测试的要求通常低于系统软件和特殊类应用软件的另一原因在于,一般类应用软件通常耗费资源较少,在一定程度上提高硬件配置不需要太大的成本,因此可以通过提高硬件配置,进而改善运行环境改善应用软件的性能。从硬件方面改

善系统性能往往更容易做到，而且很可能降低总的开发成本。

但应注意到，不管对什么类型的软件，提高硬件配置只是改善系统性能的一个手段，如果软件自身存在性能问题，再多的资源可能也不够用。例如内存泄漏问题，随着时间的增加，内存终究会被耗尽，最后导致系统崩溃。而且，若过分强调用户对硬件的投入，会给用户带来很大的负担，从而降低产品的用户满意度。因此，解决性能问题的最好方法仍是软硬件优化相结合。

还有一个值得关注的问题，就是性能测试和功能测试的关系。许多人认为性能测试是独立于功能测试的。实际上，两者有着紧密的关系，这是因为软件功能和性能是联系紧密的。若在需求分析和设计时对应用系统的功能未全面考虑或功能划分不良，就有可能影响系统性能。也就是说，软件的功能缺陷可能会带来性能问题。软件的功能与性能之间的关系给我们如下启示：

（1）应在需求分析和设计阶段确定软件功能及对功能进行细化时就考虑其对性能的影响，尽量从源头提升软件性能。

（2）功能测试可以发现性能问题。

（3）性能测试也能发现功能问题。

（4）性能测试和功能测试是紧密相连的，功能测试通常要先于性能测试执行或者同步进行。若功能测试导致对软件的修改，应有针对性地再次进行性能测试。

2. 性能测试策略实例

下面给出一些针对性能测试策略制定的实例。

案例 1　一个银行项目的性能测试策略。

测试策略如表 7-2 所示。

表 7-2　某银行项目的性能测试策略实例

产品类型	银行卡审批业务系统，使用非常繁忙，业务量每年约 200 万，属于银行领域的特殊类应用软件
项目背景	系统属于第二次重新开发，前一开发商在系统开发完成后未通过性能测试，100 个左右用户并发访问系统时数据库服务器崩溃。因此新的系统从项目启动开始，就着手制定性能测试策略
用户要求	用户高度重视性能，系统性能达标成为用户接收产品的首要条件
性能测试策略	① 从设计阶段开始进行性能测试的准备工作，主要是在系统设计时考虑优化性能的因素。由于前一开发商失利的主要原因是数据库设计不合理从而导致性能问题，故重点对数据库设计进行研究。在设计阶段完成性能测试策略的制定 ② 单元测试阶段通过测试工具对核心业务模块的并发性能进行测试 ③ 集成测试阶段进行组合业务性能测试 ④ 整个系统测试阶段都在进行性能测试，性能测试和功能测试应同步进行，对功能测试导致的修改，应再进行相关的性能测试 ⑤ 验收测试阶段时，在用户现场的投产环境进行性能测试，根据测试结果对系统运行环境进行调优，以达到较好的运行效果

案例 2 一个 OA 系统的性能测试策略。

测试策略如表 7-3 所示。

表 7-3 某 OA 系统的性能测试策略实例

产品类型	企业办公自动化系统，用户数目在 300 人以内，主要是完成信息的发布、公文流转、邮件收发等功能。该软件属于一般类应用软件，对性能的要求不高，性能测试不属于测试中的重点
项目背景	已有稳定的产品，主要是根据客户的个性化需求进行二次开发
用户要求	在 300 个用户并发使用时，各独立业务及组合业务的响应时间均不超过 2 秒
性能测试策略	① 系统测试阶段开始进行性能测试策略的制定 ② 性能测试在系统测试阶段的功能测试之后进行，主要是评估系统性能并对其进行优化 ③ 验收测试阶段在用户现场的投产环境进行性能测试，根据测试结果对系统运行环境进行调优，以达到较好的运行效果

案例 3 一个门户系统的性能测试策略。

测试策略如表 7-4 所示。

表 7-4 某门户系统的性能测试策略实例

产品类型	主要用于单位信息的发布，用户在 50 人以内。该系统属于一般类应用软件，对性能要求低
项目背景	软件运行的硬件环境较好
用户要求	没有具体的要求
性能测试策略	① 性能测试策略在系统测试的后期开始制定 ② 验收测试阶段在用户现场的投产环境进行性能测试，根据测试结果对系统运行环境进行调优，以达到较好的运行效果

可以看出，以上 3 个案例的性能测试策略有着较大的差别。通过这 3 个案例，可对性能测试策略的制定有更进一步的了解。实际上，除了应用系统的自身特点和用户对性能的关注程度外，影响性能测试策略的还包括项目的背景、软件运行环境等。因此，制定性能测试策略时应充分考虑各方面的实际因素，灵活地制定性能测试策略。

7.3.3 全面性能测试模型的使用

全面性能测试模型是针对 Web 性能测试提出的一种方法，主要目的是使 Web 性能测试更容易组织和开展。要想在性能测试中用好、用活全面性能测试模型，首先要针对具体的应用系统制定出合理的性能测试策略，同时还应注意遵循如下基本原则。

1. 最低测试成本原则

Web 性能测试本身是一种高投入的测试，若开发商（或测试外包提供商）用于测试的资金有限，不能对软件项目的全部性能指标进行严格的测试，因而在制定性能测试策略时应遵循最低测试成本原则。

最低性能测试成本的衡量标准主要指投入的测试成本能否使系统满足预先确定的性能测试目标。

2. 用例裁剪原则

全面性能测试模型主要是针对电信、金融等特殊类应用软件而提出的。这类软件的

业务重要性级别高，对系统性能要求高，因而包含的测试内容比较全面，测试用例数目较大。对于一般的应用系统，可根据系统自身的特点和用户对性能的要求，对根据全面性能测试模型设计的测试用例进行适当的裁剪，以缩短性能测试周期，节约测试成本。

3. 模型具体化原则

全面性能测试模型的使用绝不能生搬硬套，要使全面性能测试模型在 Web 性能测试或其他类型应用系统的性能测试中真正发挥作用，应根据实际软件项目的特点、软件运行环境、用户对系统性能的要求等因素制定出合理的性能测试策略，编写适当的性能测试用例，并在测试实施过程中灵活地执行测试方案。

7.4　基于 Web 的系统测试

随着网络技术和软件技术的发展，Web 应用日益广泛，Web 系统的重要性也日益突出。在这一节中将全面介绍基于 Web 的系统测试。

基于 Web 的系统测试不仅要检验系统是否能按既定的设计要求正确地完成相应功能，还要从用户的角度进行诸如可用性、安全性等方面的测试。此外，Web 系统一般较复杂，涉及的环节较多，如网络、硬件等，各组件能否很好地协同工作也是需要测试的方面。然而，Internet 和 Web 媒体的不可预见性使测试基于 Web 的系统变得困难。因此，有必要专门研究针对 Web 的系统测试内容和方法。当然，如果能够对基于 Web 的系统测试得心应手，其他类型软件的系统测试应该是不成问题的。

基于 Web 的系统测试主要可分为功能测试、性能测试、用户界面测试、兼容性测试和配置测试、安全测试和接口测试，如图 7-1 所示。其中，Web 系统的性能测试已在前面介绍过，下面对其他部分加以介绍。

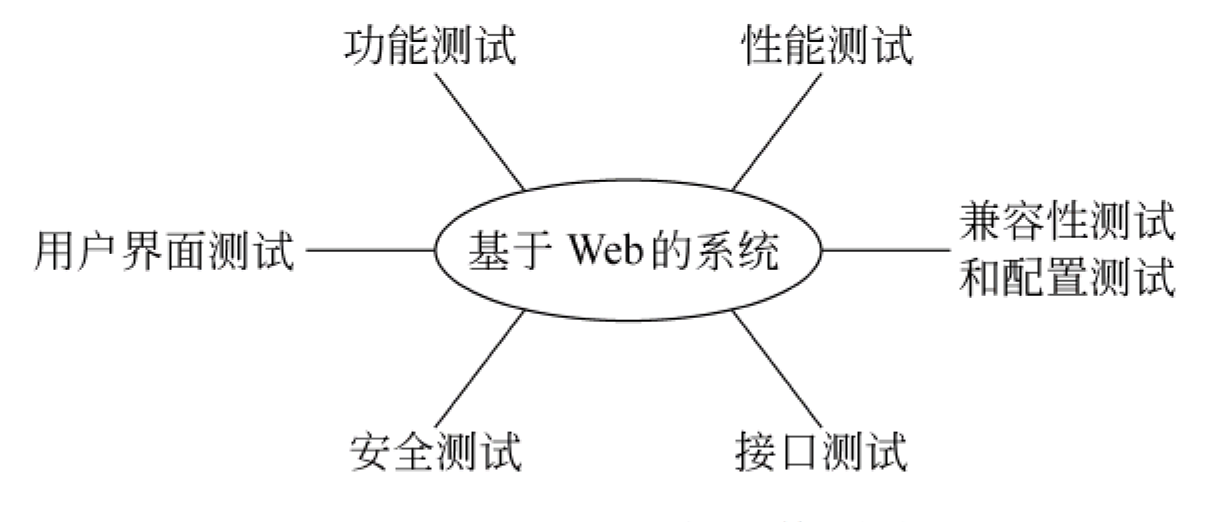


图 7-1　基于 Web 的系统测试

7.4.1　功能测试

1. 链接测试

链接是 Web 应用系统的一个主要特征，它是在页面之间切换和指导用户访问未知地址的页面的主要手段。链接测试可包括以下 3 个步骤：

① 测试所有链接是否按指示的那样确实链接到应该链接的页面。

② 测试所链接的页面是否存在。

③ 保证 Web 应用系统上没有孤立的页面。所谓孤立页面是指没有链接指向该页

面，只有知道正确的URL地址才能访问。

应在整个Web应用系统的所有页面开发完成之后进行链接测试。

链接测试可以自动进行，现在已经有许多自动检测网站链接的软件，如Xenu Link Sleuth，HTML Link Validator。

Xenu Link Sleuth是一个小巧但功能强大的检查网站死链接的免费、免安装软件，用户可以打开一个本地网页文件来检查它的链接，也可以输入任何网址来检查。它可以分别列出网站的活链接及死链接，连转向链接它都分析得一清二楚。Xenu Link Sleuth还支持多线程，可以把检查结果存储成文本文件或网页文件。

网页链接检查程序HTML Link Validator（共享软件），可以帮助用户进行ftp：//和http：//的链接检查，判断是否有无法链接的内容。

2. 表单测试

当用户通过表单提交信息的时候，都希望表单能够正常工作。如果使用表单进行在线注册，应确保提交按钮能正常工作，当注册完成后应返回注册成功的消息。如果使用表单收集配送信息，应确保程序能够正确地处理这些数据，最后能让客户收到包裹。要测试这些程序，需要验证服务器能正确地保存这些数据，而且后台运行的程序能正确地解释和使用这些信息。

当用户使用表单进行用户注册、登录、信息提交等操作时，必须测试提交操作的完整性，以校验提交给服务器的信息的正确性。例如，用户填写的出生日期与职业是否恰当，填写的所属省份与所在城市是否匹配等；如果使用默认值，还需检验默认值的正确性。如果表单只能接受指定的某些值，则也要进行测试。例如，若只能接受某些字符，测试时可以跳过这些字符，看系统是否会报错。

3. 对数据校验的测试

如果Web系统根据业务规则需要对用户的输入进行校验，则需要保证这些校验功能能够正常使用。

例如，省份的字段可以用一个有效列表进行校验。在这种情况下，需要验证列表完整而且程序正确调用了该列表，可在列表中添加一个测试值，确定系统是否能够接受这个测试值。

在测试表单时，对数据校验的测试和表单测试可能会有一些重复。

表单测试和对数据校验的测试可采用的策略是，第一个完整的系统版本采用手动检查，同时形成WinRunner或QTP等功能测试工具的脚本，进行回归测试主要依靠测试工具自动回放脚本完成测试。

4. Cookies 测试

Cookies通常用来存储用户信息和用户在某应用系统的操作。当一个用户使用Cookies访问了某一个应用系统时，Web服务器将发送关于用户的信息，把该信息以Cookies的形式存储在客户端计算机上，这可用来创建动态和自定义页面或者存储登录等信息。

如果Web应用系统使用了Cookies，就必须检查Cookies是否能正常工作。测试的

内容可包括 Cookies 是否起作用、是否按预定的时间进行保存、刷新对 Cookies 有什么影响等。如果在 Cookies 中保存了注册信息,应确认该 Cookies 能够正常工作而且已对这些信息进行了加密。如果使用 Cookies 来统计次数,需要验证次数累计是否正确。

如何进行 Cookies 测试呢? 可采用上文提到的方法进行黑盒测试,也可采用查看 Cookies 的工具软件进行,如 IECookiesView V1.70 和 Cookies Manager V1.1。

IECookiesView V1.70 可以帮助用户搜寻并显示出自己计算机中所有的 Cookies 档案的数据,包括是哪一个网站写入 Cookies 的,内容有什么,写入的时间日期及此 Cookies 的有效期限等资料,以防止用户隐私的泄露。此软件只对 IE 浏览器的 Cookies 有效。

Cookies Manager V1.1 则可帮助我们查看和管理自己硬盘中的 Cookies。

当然,用查看 Cookies 的工具只能辅助对 Cookies 进行测试。

5. 数据库测试

在 Web 应用技术中,数据库起着重要的作用。数据库为 Web 应用系统的管理、运行、查询和实现用户对数据存储的请求等提供了空间。在 Web 应用中,最常用的数据库类型是关系型数据库,可以使用 SQL 对信息进行处理。

在使用了数据库的 Web 应用系统中,一般情况下可能发生两种错误,即数据一致性错误和输出错误。数据一致性错误主要是由于用户提交的表单信息不正确而造成的,而输出错误主要是由于网络速度或程序设计问题等引起的。针对这两种情况,可分别进行测试。

6. 对应用程序特定功能需求的测试

最重要的是,测试人员需要对应用程序特定的功能需求进行验证,尝试用户可能进行的所有操作,如下订单、更改订单、取消订单、核对订单状态、在货物发送之前更改送货信息、在线支付等。因为这是用户使用网站的原因,一定要确认网站能够完成广告宣传中的所有功能。

7. 设计语言测试

Web 设计语言版本的差异可以引起客户端或服务器端严重的问题,如使用哪种版本的 HTML 等。当在分布式环境中开发时,开发人员都不在一起,这个问题就显得尤为重要。除了 HTML 的版本问题外,不同的脚本语言,例如 Java,JavaScript, ActiveX,VBScript 或 Perl 等也需要进行验证。

7.4.2 用户界面测试

1. 导航测试

导航描述了用户在一个页面内操作的方式(在不同的用户接口控制之间,例如按钮、对话框、列表和窗口等,或在不同的连接页面之间)。

通过考虑下列问题,可以决定一个 Web 应用系统是否易于导航:

- 导航是否直观;
- Web 系统的主要部分是否可通过主页存取;
- Web 系统是否需要站点地图、搜索引擎或其他导航帮助。

2. 图形测试

在Web应用系统中，适当的图片和动画既能起到广告宣传的作用，又能美化页面。一个Web应用系统的图形可以包括图片、动画、边框、颜色、字体、背景、按钮等。图形测试的内容如下：

- 确保图形有明确的用途，图片或动画切忌乱堆在一起，以免浪费传输时间。Web应用系统的图片应能清楚地说明某件事情，一般都链接到某个具体的页面。
- 所有页面字体的风格是否一致。
- 背景颜色与字体颜色和前景颜色是否匹配。
- 图片的大小和质量也是一个很重要的因素，一般采用JPG或GIF压缩，最好能使图片的大小减小到30KB以下。
- 文字回绕是否正确。不要因为使用图片而使窗口和段落排列古怪或者出现孤行。

3. 内容测试

内容测试用来检验Web应用系统提供信息的正确性、准确性和相关性。

信息的正确性是指信息的来源是可靠的，例如在商品价格列表中，错误的价格可能引起财政问题甚至导致法律纠纷。信息的准确性是指是否存在语法或拼写错误，这种测试通常使用一些文字处理软件来进行，例如使用Microsoft Word的“拼音与语法检查”功能。信息的相关性是指是否在当前页面可以找到与当前浏览信息相关的信息列表或入口，也就是一般Web站点中的所谓“相关文章列表”。

4. 表格测试

需要验证表格是否设置正确。例如，表格中每一栏的宽度是否足够宽，单元格里的文字是否都有折行，是否有因为某一格的内容太多而将整行的内容拉长，这些都是需要考虑的问题。

5. 整体界面测试

整体界面是指整个Web应用系统的页面结构。例如，当用户浏览Web应用系统时是否感到舒适，是否凭直觉就知道要查找的信息在何处，整个Web应用系统的设计风格是否一致等。

对整体界面的测试过程，其实是一个对最终用户进行调查的过程。一般Web应用系统采用在主页上做一个调查问卷的形式，得到最终用户的反馈信息。

7.4.3 兼容性测试和配置测试

1. 平台测试

操作系统的类型很多，最常见的有Windows，UNIX，Macintosh，Linux等。Web应用系统的最终用户究竟使用哪一种操作系统，取决于用户系统的配置。这样，就可能产生兼容性问题，同一个应用可能在某些操作系统下能正常运行，但在另外的操作系统下可能会运行失败。

因此，在Web系统发布之前，需要在各种操作系统下对Web系统进行兼容性测试。

2. 浏览器测试

浏览器是 Web 客户端最核心的构件,来自不同厂商的浏览器对 Java,JavaScript,ActiveX,Plug-ins 或不同的 HTML 规格有不同的支持。例如,ActiveX 是 Microsoft 的产品,是为 Internet Explorer 设计的;JavaScript 是 Netscape 的产品,Java 是 Sun 的产品等。另外,框架和层次结构风格在不同的浏览器中可能有不同的显示,甚至根本不显示。不同的浏览器对安全性和 Java 的设置也不相同。

测试浏览器兼容性的一个方法是创建一个兼容性矩阵,在这个矩阵中测试不同厂商、不同版本的浏览器对某些构件和设置的适应性。

3. 分辨率测试

测试页面在 640×400,600×800 或 1024×768 的分辨率模式下是否能够显示正常,字体是否太小或太大,文本和图片是否对齐。

4. Modem 连接速率测试

应测试 Web 用户通过不同厂商生产的 Modem 上网时的连接速度,即测试用户通过 Modem 链接到某页面或下载某资料时所需要的时间。测试应在平常的使用环境下进行。

5. 打印机测试

有时,在屏幕上显示的图片和文本的对齐方式与打印出来的不一样,这就需要验证网页打印是否正常。

6. 组合测试

最后需要进行组合测试,也就是根据 Web 系统可能的使用环境对各种软硬件配置进行组合测试。

7.4.4　安全测试

即使站点不接受信用卡支付,安全问题也是非常重要的。Web 站点收集的用户资料只能在公司内部使用,以确保客户在进行交易时的绝对安全;还应保证公司的商业机密不被竞争对手或谈判客户获取。

1. 目录设置测试

Web 安全的第一步就是正确地设置目录。每个目录下应该有 index.html 或 main.html 页面,这样就不会显示该目录下的所有内容,确保公司商业机密或客户信息不被泄露。

2. SSL 测试

很多站点使用 SSL(Security Socket Layer,安全套接字)进行安全传送。SSL 是一种加密通信协议,使用 SSL 可以防止传送的内容及用户密码等信息被黑客获取。

用户确定自己进入一个 SSL 站点是因为浏览器出现了警告消息,而且在地址栏中的 HTTP 变成 HTTPS。

如果开发部门使用了 SSL,测试人员就需要确定是否有相应的替代页面(适用于 3.0 以下版本的浏览器,这些浏览器不支持 SSL);还应确认是否有连接时间限制,以及超过限

制时间后会出现什么情况。当用户进入或离开安全站点时,应确认是否有相应的提示信息。

3. 登录测试

有些站点需要用户进行登录,以验证他们的身份。对登录进行测试时,可考虑如下方面:

- 系统能否阻止非法的用户名或口令登录,而对有效的登录予以通过。
- 用户登录在某个时段若有次数限制,该限制是否有效。
- 若限制从某些 IP 地址登录,该机制是否有效。
- 如果对同一用户登录有输入密码的次数限制,该机制是否有效。
- 对口令选择的限制的测试。
- 是否可以不登录而直接浏览某个页面。
- 对 Web 应用系统的超时限制(如用户登录后在一定时间内没有点击任何页面,则需要重新登录才能正常使用)的测试。

4. 日志文件测试

在后台,应注意验证服务器日志工作是否正常,可考虑如下方面:

- 是否记录了所有的事务处理;
- 是否记录了失败的注册企图;
- 是否记录了被盗信用卡的使用。
- 是否在每次事务完成的时候都进行了保存。
- 是否记录了 IP 地址。
- 是否记录了用户名。

5. 脚本语言安全性测试

脚本语言是常见的安全隐患。每种脚本语言的细节均有所不同,有些脚本允许访问根目录,其他只允许访问邮件服务器。经验丰富的黑客可以将服务器用户名和口令发送给他们自己,进而找出该站点使用了哪些脚本语言,并研究该语言的缺陷从而对站点进行攻击。测试时应判断是否可能发生此种安全隐患。

此外,测试人员还要需要测试没有经过授权,就不能在服务器端放置和编辑脚本等问题。

7.4.5 接口测试

通常情况下,Web 站点不是孤立的。Web 站点可能会与外部服务器通信以协同工作,例如请求数据、验证数据或提交订单。

1. 服务器接口测试

第一个需要测试的接口是浏览器与服务器的接口。测试人员提交事务,然后查看服务器记录,并验证在浏览器上看到的正是服务器上发生的。测试人员还可以查询数据库,确认事务数据已正确保存。

这种测试可以归到功能测试中的表单测试和数据校验测试中。

2. 外部接口测试

有些 Web 系统有外部接口。例如，网上商店可能要实时验证信用卡数据以减少欺诈行为的发生。测试的时候，应使用 Web 接口发送一些事务数据，分别对有效信用卡、无效信用卡和被盗信用卡进行验证。

通常，测试人员需要确认软件能够处理外部服务器返回的所有可能的消息。

3. 对异常处理的测试

应在充分理解用户需求的基础上，尽量全面地列出各种异常情况。例如：

- 在处理过程中中断事务；
- 中断用户到服务器的网络连接；
- 中断 Web 服务器到信用卡验证服务器的连接。

测试在这些情况下，系统能否正确地处理发生的异常。

7.5　回归测试

回归测试(Regression Testing)是在系统测试阶段常要面临的工作，下面简要介绍之。

7.5.1　回归测试的概念

软件在其生命周期中时常面临变更，软件的变更可能源于在测试中发现了错误并进行了修改，也可能是因为在集成或维护阶段加入了新的模块。

当软件中所含错误被发现时，如果错误跟踪与管理系统不够完善，就可能会遗漏对这些错误的修改；而开发者对错误理解的不够透彻，也可能导致所做的修改只修正了错误的外在表现，而没有修复错误本身，从而造成修改失败；修改还有可能产生副作用从而导致软件未被修改的部分产生新的问题，使本来工作正常的功能产生错误。

同样，在有新代码加入软件的时候，除了新加入的代码中有可能含有错误外，新代码还有可能对原有的代码带来影响。

为排除出现上述问题的可能，应进行回归测试。简单地说，进行回归测试的目的是验证修改的正确性及修改是否对未被修改部分造成不良影响。使用回归测试的时机如图 7-2 所示。

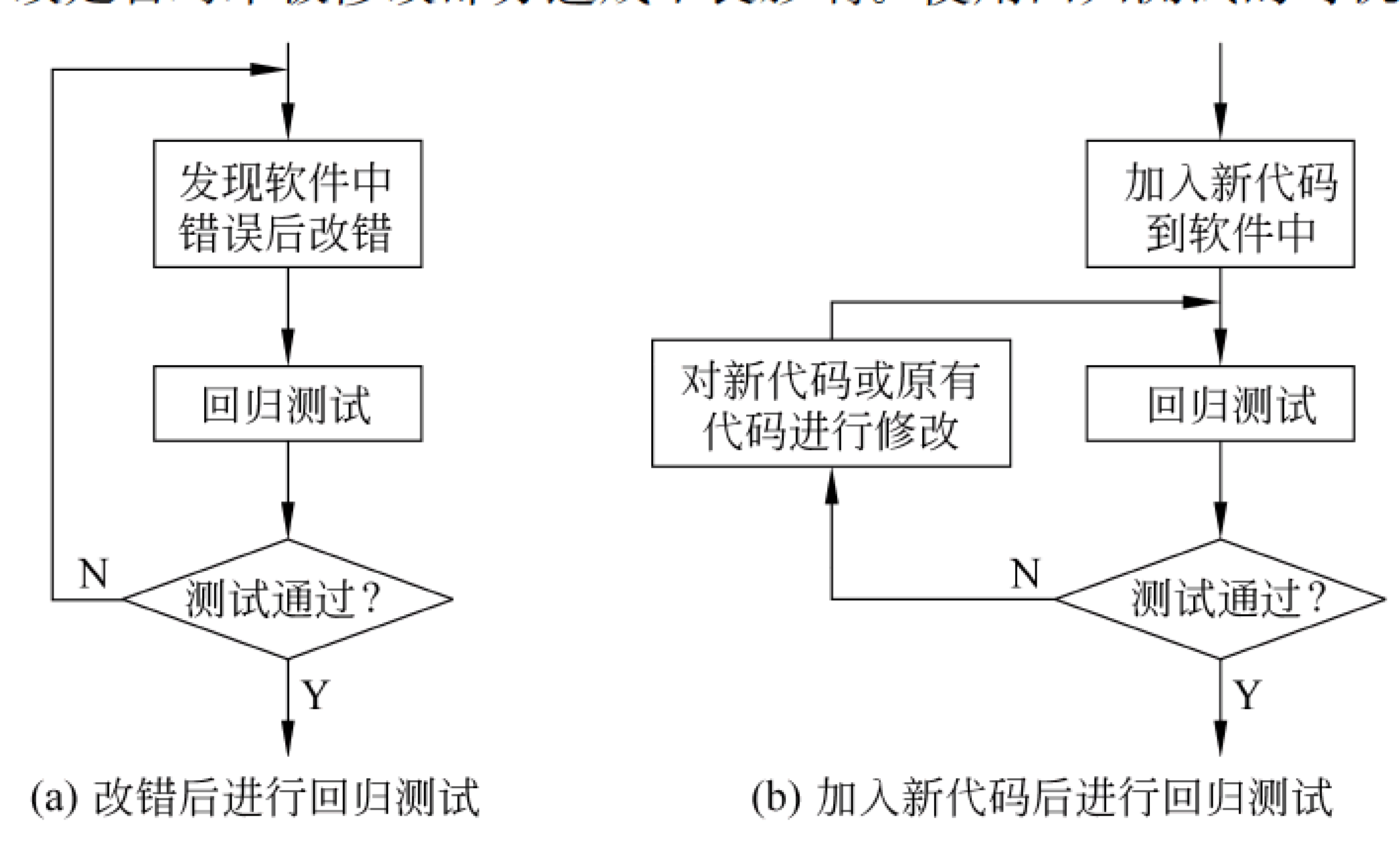


图 7-2　回归测试的使用时机

在实际测试工作中,回归测试需要反复进行,给人带来繁琐的感觉。因此,需要通过自动测试来减轻回归测试的人工工作量和提高回归测试的效率。为了支持多种回归测试策略,自动测试工具应该是通用的和灵活的,以满足达到不同回归测试目标的要求。

回归测试工具往往是与功能测试工具融合在一起的。关于回归测试工具,将在第9章中介绍。

7.5.2 回归测试策略

在集成测试、系统测试甚至单元测试阶段都会进行多次回归测试。在渐进和快速迭代开发中,新版本的连续发布使回归测试进行得更加频繁;而在极限编程方法中,更是要求每天都进行若干次回归测试。因此,选择正确的回归测试策略提高回归测试效率是很有必要的。

测试组在实施测试的过程中会将所开发的测试用例保存到测试用例库中,并对其进行维护和管理。当得到一个软件的基线(Baseline)版本时,用于基线版本的所有测试用例就形成了基线测试用例库。在进行回归测试的时候,可根据回归测试策略从基线测试用例库中提取合适的测试用例进行回归测试。保存在基线测试用例库中的测试用例可能是自动测试脚本,也有可能是测试用例的手工实现过程。

1. 测试用例库的维护

随着软件的改变,测试用例库中的一些测试用例可能会失去针对性和有效性,还有一些测试用例将完全不能运行,必须删除测试用例库中这一些测试用例。

同时,被修改的或新增添的软件功能,仅仅靠重新运行以前的测试用例并不足以揭示其中的问题,有必要追加新的测试用例来测试这些新的功能或特征。因此,测试用例库的维护工作还应包括开发新测试用例。

此外,随着项目的进展,测试用例库中的用例会不断增加,其中会出现一些对输入或运行状态十分敏感的测试用例。这些测试不容易重复且结果难以控制,会影响回归测试的效率,需要进行改进,使其达到可重复和可控制的要求。

2. 测试用例的选择方法

在计划回归测试时,常用的选择测试用例的方法如下。

(1) 再测试全部用例

选择基线测试用例库中的全部测试用例组成回归测试包,这是一种比较安全的方法。再测试全部用例具有最低的遗漏回归错误的风险,但测试成本最高。全部再测试几乎可以应用到所有情况,基本上不需要进行分析和重新开发。但是,随着开发工作的进展,测试用例不断增多,重复原先所有的测试将带来非常大的工作量和高昂的成本。

(2) 基于风险选择测试

可以基于一定的风险标准从基线测试用例库中选择回归测试包。例如,首先运行最重要的、关键的测试用例,而跳过那些非关键的、优先级别低的或者高稳定的测试用例。

(3) 基于操作剖面选择测试

若基线测试用例库的测试用例是基于软件操作剖面开发的,测试用例的分布情况反

映了系统的实际使用情况，则在回归测试中可优先选择那些针对最重要或最频繁使用功能的测试用例，释放和缓解最高级别的风险，这有助于尽早发现那些对可靠性有最大影响的故障。此方法可以在给定的预算下最有效地提高系统可靠性。

7.6　系统测试步骤

为了成功地进行系统测试，应遵循一定的步骤。系统测试的步骤如下：

① 编制系统测试计划。

② 设计系统测试用例。

③ 审批系统测试计划。

④ 执行系统测试。

⑤ 分析系统测试结果，提交测试报告，并改进系统。

1. 系统测试计划的编制

根据软件测试 V 模型，系统测试计划的编制可在需求分析完成之后就开始进行（甚至可以在需求分析过程中进行）。在计划的编制中应主要完成如下任务：

（1）系统测试内容和测试类型的确定。应根据软件项目的特点、开发的预期要求、软件的运行环境等多方面因素，从前面介绍的诸多系统测试类型中选择若干种（当然最好是将每种类型都执行到），确定每种系统测试类型的具体策略，以及对它们的执行顺序，且将能用测试工具完成的测试部分确定下来。

（2）确定系统测试进度。

（3）确定系统测试小组的人员分工。

（4）建立系统测试环境，选择（或自行开发）和安装系统测试工具。

2. 系统测试用例的设计

应根据每种系统测试类型的具体策略设计系统测试用例，能够用测试工具执行的用例应当对其脚本化。

3. 系统测试计划的审批

应对系统测试计划进行评审，对测试计划的框架和细节进行讨论，有修改建议时应及时反馈并文档化。若系统测试计划获得批准，应有一个正式签名的过程。

4. 系统测试的执行

根据系统测试计划和事先确定的系统测试用例执行系统测试，并详细记录测试过程中发现的问题。系统测试不是一个一蹴而就的过程，可能会有多次的回归测试。

5. 分析系统测试结果，提交测试报告并改进系统

对系统测试的结果进行细致、深入的分析，根据系统测试的全过程提交系统测试报告。并根据系统测试中发现的问题对应用系统进行必要的改进，以对其进行优化。

7.7 验收测试

7.7.1 验收测试的概念

验收测试是软件正式交付使用之前的最后一道测试工序。软件开发的最终目的是满足用户的需要,所以验收测试是以用户为主进行的测试,软件开发人员、SQA 人员也应参与。

验收测试检验最终软件产品与用户预期的需求是否一致,决定软件是否可被用户接受。验收测试应着重考虑软件是否满足合同规定的所有功能、性能及其他特性。

验收测试的结果有两种可能:一种是功能、性能等各项指标满足软件需求规格说明的要求,用户可以接受;另一种是软件不满足需求规格说明的要求,用户无法接受。项目进行到这个阶段才发现严重错误和偏差一般很难在预定的工期内改正,因此必须与用户协商,寻求一个妥善解决问题的方法。

7.7.2 验收测试的策略

验收测试的策略通常有正式验收、非正式验收或 α 测试、β 测试 4 种,选择验收测试的策略通常建立在合同需求、组织、公司标准,以及应用领域要求的基础之上。

1. 正式验收

正式验收的过程如下:

① 软件需求分析。了解软件功能和性能要求、软硬件环境要求等,特别需要了解软件的质量要求和验收要求。

② 编制“验收测试计划”和“项目验收准则”。根据软件需求和验收要求编制测试计划,制定需测试的测试项,制定测试策略及验收通过准则,并经过客户参与的计划评审。

③ 设计测试用例。根据“验收测试计划”和“项目验收准则”编制测试用例,并经过评审。

④ 搭建测试环境。建立测试的软、硬件及网络环境(不应是开发环境,可在委托客户提供的环境中进行测试)。

⑤ 实施测试。测试并记录测试结果。

⑥ 分析测试结果。根据验收通过准则分析测试结果,对测试进行评价,并确认验收是否通过。

⑦ 编制并提交测试报告。根据测试结果编制缺陷报告和验收测试报告,并提交客户。

正式验收测试是一个需要严格规划和组织的过程,它通常是系统测试的延续,选择的测试用例应该是系统测试中所执行测试用例的子集。在很多组织中,正式验收测试是完全自动执行的。

正式验收测试可由开发小组(或其独立的测试小组)与最终用户组织的代表来执行,也可能完全由最终用户团队执行,或者由最终用户团队选择人员组成一个客观公正的小组来执行。

正式验收的主要优点是测试可以自动执行，支持回归测试，并可以对测试过程进行评测和监测；不足之处主要是要求大量的资源和周密的计划，测试可能是系统测试的再次实施，测试成本有一定的浪费。

2. 非正式验收

在非正式验收中，测试过程不像正式验收测试那样严格，显得比较主观。非正式验收测试也应事先确定测试项，但这是由各测试员自行决定的。

大多数情况下，非正式验收是由最终用户执行的。

非正式验收与正式验收相比，可以发现更多意料之外的软件缺陷(例如用户操作方式有误时软件不能恰当处理)，但由于缺乏严格的计划组织，非正式测试发现的错误往往是有限的。

关于 α 测试和 β 测试可参见第 2.2.2 小节。

7.8 小结

确认测试、系统测试和验收测试都属于收尾阶段的测试活动，用来保证软件产品达到了需求规格说明书中关于软件功能、性能及其他特性的各项要求，并能最终满足用户的各项需求，能为用户所接受。

这 3 种测试活动的工作有较大的相似之处，在业界也往往没有被严格区分，但我们仍应注意它们之间的区别：确认测试可以认为是系统测试的一部分，以功能测试为主；系统测试则强调将软件与系统中其他要素集成起来，测试其能否正常工作，系统测试不仅关注功能的实现，还关注性能及其他重要特性(如安全性、兼容性、界面友好性等)是否达到预期的要求；验收测试则是以用户为主进行的测试活动，是软件交付使用之前的最后一道测试活动。

性能测试是系统测试中的重要环节，应掌握性能测试的几种主要类型及性能测试的策略。

基于 Web 的系统测试比一般应用的系统测试更为复杂，可通过它了解系统测试的各个侧面。

习 题

1. 确认测试的任务是什么？确认测试主要包括哪些具体活动？
2. 性能测试主要包括哪几种类型？分别解释它们的含义。
3. 全面性能测试模型的含义是什么？如何在具体项目的性能测试中合理地使用它？
4. 安全性测试的目的是什么？
5. 健壮性测试的含义是什么？它包括哪几种测试？
6. 如何衡量软件产品的可靠性？
7. 配置测试和兼容性测试的区别是什么？
8. 用户界面测试、文档测试及安装测试的目的分别是什么？

9．基于 Web 的系统测试主要包括哪些方面？

10．系统测试与集成测试的区别是什么？

11．何时应进行回归测试？回归测试的作用是什么？

12．在不断进行回归测试的过程中，如何对测试用例库进行维护？

13．在计划回归测试时，选择测试用例的方法有哪些？试分别解释之。

14．常用的验收测试策略有哪几种？各有何优缺点？

15．确认测试、系统测试和验收测试三者的主要区别是什么？应分别由何种人员来执行？

第 8 章

面向对象软件的测试

本章要点：

- 面向对象的基本概念。
- 面向对象方法的优缺点。
- 面向对象软件的特点及其对软件测试的影响。
- 面向对象软件的测试模型。
- 面向对象软件的单元测试。
- 面向对象软件的集成测试。

面向对象是目前主流的软件开发方法，占据了软件开发的绝大部分领地。面向对象方法有着与传统开发方法完全不同的思维视角，这使得面向对象软件也与传统软件有着诸多不同的特性，这些特性使传统的测试方法在面向对象软件的测试中基本难以适用。本章介绍面向对象方法的一些基本概念和面向对象软件测试的重要性，以及面向对象软件的特点对软件测试的影响，最后通过面向对象软件的测试模型详细阐述了面向对象软件测试各部分的概念和实施策略。

8.1　面向对象开发方法概述

面向对象(Object Oriented，OO)开发是 20 世纪 90 年代以来软件开发方法的主流。面向对象的概念和应用已超越程序设计，扩展到很多领域，例如数据库系统、交互式界面、应用结构、应用平台、分布式系统、网络管理结构、CAD 技术、人工智能等领域。

8.1.1　面向过程开发方法的不足

面向过程的开发方法关注解决问题的过程，而不关心问题领域的实体。它对过程进行抽象，采用“自顶向下，逐步求精”的思路对要实现的功能模块进行逐层分解，从而简化问题的实现。

面向过程方法是一种成熟的开发方法，然而在大型软件的开发中，面向过程方法则显得力不从心，主要是因为它存在如下不足之处。

1. 软件重用性差

软件重用是软件工程追求的目标之一。所谓软件重用，是指在不同的软件开发过程中重复使用相同或相似软件元素的过程。这里所说的软件元素包括源代码、测试用例、设计文档、设计过程、需求分析文档等，可将这些软件元素称为软部件。

由于面向过程的开发是以过程为中心的，没有引入对象和类的概念，更没有继承的机制，而且封装机制也不够强大，所以面向过程方法开发出的软件的部件重用性较差。

2. 安全性差

封装机制不如面向对象方法强大，故安全性不如面向对象软件，这一点在面向对象方法的介绍中可以看出。

3. 软件稳定性和可维护性差

用面向过程方法开发大型的、复杂的软件时，开发初期难以明确全部需求，在开发过程中常常面临需求变更的情况，此时面向过程软件就会表现出较差的适应性和稳定性。这是因为面向过程的方法是围绕实现处理功能的“过程”来构造系统的。然而，用户需求的变化大部分是针对功能的，因此这种变化对于基于过程的设计来说是灾难性的。用户需求的变化往往造成系统结构的较大变化，为实现这些变化往往需要付出极大的代价，即使如此还可能不能满足用户的全部需求。

此外，对面向过程软件的维护成本也比较高，其原因是软件的可修改性、可测试性差，导致维护困难。

8.1.2 面向对象的基本概念

关于面向对象，有许多不同的观点。Coad 和 Yourdon 曾给出如下定义：

面向对象(object oriented)＝对象(object)＋类(class)＋继承(inheritance)
＋消息通信(message communication)

也就是说，若某软件的开发过程中用到了以上 4 个概念(或机制)，则该开发过程是面向对象的。

下面对面向对象方法中最基本的概念进行简要的介绍。

1. 对象

对象是现实世界中存在的一个事物。对象可以是具体的，例如一张桌子；也可以是抽象的，例如一个开发项目或一个计划。

在面向对象开发中，对象就是模块，它是把数据结构和操作这些数据的方法紧密地结合在一起而构成的模块。

2. 类和实例

具有相同特征和行为的所有对象构成一个类，属于某个类的对象称为该类的实例(instance)。例如，张三、李四、王五是不同的对象，但他们有相同的特征和行为，因为他们都是学生，故“学生”是从张三、李四、王五抽象出来的类，张三、李四、王五这几个对象则是“学生”类的实例。

类抽象地描述了属于该类的全部对象的属性(用数据结构表示)和操作(也称为服务

或方法,在 C ++ 语言中即成员函数)。可将类看作一个抽象数据类型(Abstract Data Type,ADT)的实现。

3. 继承

继承是子类(也称派生类)自动共享其父类和祖先类属性和操作(即共性部分)的一种机制。子类在继承共性部分的基础上,还可以增加自身特有的属性和操作。当然,若不合适,子类也可放弃对父类和祖先类中某些特性的继承。

例如,轿车、货车、摩托车、救护车是不同种类的车辆,分别对应一个类,但通过分析发现,它们具有共性——都是机动车,故可建立一个父类——机动车类;轿车类、货车类、摩托车类、救护车类则是机动车类的子类。建立子类、父类的继承机制后,则可把这 4 个子类的共性部分放在父类中定义,避免了类中的重复定义。

子类只继承一个父类的属性和操作,称为单重继承;子类若继承了多个父类的属性和操作,则称为多重继承。例如,党员和研究生作为两个父类,研究生党员继承了两者的共性,就属于多重继承。

继承是面向对象开发方法独有的机制。继承机制的优点如下:

(1) 使共有的属性和操作能够共享,避免了类中的重复定义,增加了代码的可重用性。

(2) 由于代码重用性高,因而缩短了代码的总长度,使程序简短、结构清晰、易于理解。

(3) 对于具有继承关系的父类和多个子类的相同部分的修改,只需在父类中进行即可。

4. 消息

两个对象之间的通信单元称为消息,它是要求接收消息的对象执行类中定义的某些操作的规格说明。消息机制类似于面向过程开发中的函数调用。

5. 封装

封装(encapsulation)也可以理解为信息隐藏。对象是封装的最基本单位,类定义将其说明(用户可见的外部接口)与实现(用户不可见的内部实现)显式地分开,其内部实现按照具体定义的作用域提供保护。

虽然面向过程开发方法也有封装,但面向对象的封装比面向过程开发中的封装更为强大、有力。

封装机制具有如下优点:

(1) 简化了对对象的使用。外部程序仅通过接口访问对象,而不必知道对象内部的具体实现。

(2) 为软件模块的安全性提供了强有力的保障,因为对象内部数据结构是不能被外界访问的。

(3) 减少了类之间的相互依赖,使程序结构更为紧凑、清晰,提高了软件部件的重用性,使得对软件的修改、测试、维护等工作更易于进行。

6. 抽象

抽象有两方面的意义。一方面，尽管问题域中的事物是复杂的，但分析人员并不需要了解和描述它们的一切，只需要分析研究其中与系统目标有关的事物及其本质特征；另一方面，通过舍弃个体事物在细节上的差异，抽取其共同特征而得到一批事物的抽象概念。

与面向过程仅支持过程抽象不同，面向对象方法中的抽象原则包括过程抽象和数据抽象两个方面。

过程抽象是指任何一个完成确定功能的操作序列，其使用者都可以把它看作一个单一的实体，尽管实际上它可能是由一系列更低级的操作完成的。

数据抽象根据施加于数据之上的操作来定义数据类型，并限定数据的值只能由这些操作来修改和查看。数据抽象是面向对象开发方法的核心原则。它强调把属性（数据结构）和操作（服务）结合为一个不可分的单位（即对象），对象的外部只需要知道它做什么，而不必知道它如何做。数据抽象是通过封装机制实现的。

7. 多态性

一个操作在不同的类中可以有不同的实现方式，称为多态性（polymophism）。因此，属于不同类的对象，收到同一消息却可以产生不同的结果。

多态性增强了软件的灵活性、重用性和可维护性。

8.1.3 面向对象开发方法的优点

通过以上叙述，面向对象开发的优点归纳如下。

1. 软件重用性高

由于面向对象方法有强大的封装机制和继承机制，使得用该方法开发出来的软件部件重用性高。

重用性高带来了两个好处，一是能提高开发的效率；另一个是当重用软部件时，由于使用的是已被测试过的部件，因而使系统出错的可能性大大减小。

2. 安全性高

由于对软件中的基本单位——对象进行了封装，对象的内部数据结构不能被外界访问，提高了软件模块的安全性。

3. 软件可维护性高

在既定的继承结构中，若需对类进行修改时，共性部分的修改可以仅在父类中进行。封装机制也使得对象之间的依赖性减少，便于对某个类的修改、测试和维护。

4. 容易扩展

利用面向对象开发中的继承机制，能清晰地定义和使用模块（即对象），即使模块的功能不完善，允许对它们进行扩展而对其他模块的改动很少。这使系统更灵活、更容易扩展，而且费用更低。

5. 适用于大型、复杂项目

面向对象软件在可重用性、可维护性方面的优点使得它特别适合于大型的、复杂的软

件项目，例如前期需求不明确的项目，或采用原型方法开发的项目，它对需求变更的适应性远远强于面向过程的开发方法。

8.1.4　统一建模语言 UML 简介

统一建模语言(Unified Modeling Language，UML)是一种定义良好、易于表达、功能强大且普遍适用的建模语言，为面向对象软件提供了强大的建模工具。它溶入了软件工程领域的新思想、新方法和新技术。它的作用域不仅限于支持面向对象的分析与设计，还支持从需求分析开始的软件开发的全过程。目前，已有一些基于 UML 的可视化建模工具，如 IBM Rational 公司的 Rose。

UML 是一种图形化建模语言，它使用不同类型的图从不同的角度和抽象层次描述系统模型。UML 中的一些图形可作为面向对象软件测试的依据。事实上，基于 UML 的测试是软件测试中的研究热点，并由此产生很多实验性工具。

因此，在介绍面向对象软件测试之前，先简要地介绍 UML 的图形表示法。

UML 主要提供了以下 5 类图用于面向对象建模。

1. 用例图(Use Case Diagram)

用例图展现了一组用例和参与者，以及它们之间的关系。它在一个较高的抽象层次上从外部执行者的角度描述系统功能。

2. 静态图(Static Diagram)

包括类图、对象图和包图。其中，类图不仅说明了各个类的内部结构，即类的属性和操作，而且描述了类之间泛化(即继承)、聚集、关联、依赖等静态关系。对象图是类图的实例化表示，几乎使用与类图完全相同的标识，一个对象图即类图的一个实例。由于对象存在生命周期，因此对象图只能在面向对象系统的某一个时间段内存在。包由包或类组成，表示包与包之间的关系。包图用于描述系统的分层结构。

3. 行为图(Behavior Diagram)

行为图用于描述系统的动态模型和组成对象间的交互关系，包括状态图和活动图。其中，状态图(State Diagram)作为类图的补充，用来描述一个特定对象的所有可能状态，以及引起状态转移的事件，表示单个对象在其生命周期中的全部行为。状态图展现了一个状态机，属于系统的动态视图。由于状态图强调单个对象行为的事件顺序，因而它对于类测试有着重要的意义。活动图(Activity Diagram)描述满足用例要求所要进行的活动，以及活动间的约束关系，有利于识别并行活动。

4. 交互图(Interactive Diagram)

交互图用于描述对象间的交互关系，包括顺序图和合作图。其中，顺序图(Sequence Diagram)显示对象之间的动态交互关系，着重于体现对象之间消息传递的时间顺序；合作图(Collaboration Diagram)描述相互合作的对象间的交互关系和链接关系。顺序图和合作图都是描述对象之间的交互关系，但两者的侧重点不同，顺序图着重体现对象间交互的时间顺序，而合作图则着重体现交互对象间的静态链接关系。在使用交互图时，若强调时间和顺序，应使用顺序图；如果强调上下级关系，宜选择合作图。

5. 实现图(Implementation Diagram)

包括构件图和配置图。其中,构件图描述代码部件的物理结构及各部件之间的依赖关系。一个部件可能是一个资源代码部件、一个二进制部件或一个可执行部件,它包含逻辑类或实现类的有关信息。构件图有助于分析和理解部件之间的相互影响程度。配置图定义了系统中软硬件的物理体系结构。它可以显示实际的计算机和设备(用节点表示)及它们之间的连接关系,也可显示连接的类型及部件之间的依赖性。在节点内部,放置可执行部件和对象以显示节点与可执行软件单元的对应关系。

若将 UML 的建模机制分为静态建模机制和动态建模机制两类,应用例图、类图、包图、对象图、构件图和配置图 6 种图形,属于 UML 的静态建模机制,而状态图、活动图、顺序图和合作图 4 种图形属于 UML 的动态建模机制。

8.2 面向对象软件测试概述

8.2.1 面向对象软件测试的重要性

正是由于面向对象开发有如此多的优秀特性,使得它被看成是解决软件危机的新兴技术,已占据了软件开发的绝大部分领地。面向对象开发有着更规范的编程风格,更好的系统结构,极大地优化了数据使用的安全性,提高了软件部件的重用性和可维护性。那么,面向对象软件是否还需要进行严格的测试?测试对于面向对象软件的成功开发还有意义吗?

事实上,测试对于面向对象软件的开发有着极为重要的作用,面向对象软件的质量在很大程度上取决于测试工作,主要原因如下:

(1) 尽管面向对象开发方法提供了一系列可使人们以更高的效率开发出高质量软件的机制,如封装、继承、多态等,但最终软件的质量如何,还与开发过程中技术实施的具体情况有关,与开发人员的水平有关。面向对象的开发与面向过程的开发一样,都是主要是由人来完成的,错误往往是不可避免的,故需要严格测试。

(2) 面向对象软件部件的重用率高,若不尽早进行严格的测试,错误的繁衍将给后期测试活动及软件维护带来极大的麻烦。

(3) 面向对象开发与面向过程开发有很大的不同,主要表现在它引入了一些新的机制,如继承、封装、多态等,这些机制的使用给面向对象软件带来了新的出错的可能性,使得测试的侧重点发生较大的变化。也就是说,应重点针对这些机制的使用对软件进行测试,若忽略这些方面测试的重要性,面向对象软件的质量是得不到保证的。继承、封装、多态等机制对于面向对象软件测试的影响将在 8.2.2 小节中描述。

8.2.2 面向对象软件的特点及其对软件测试的影响

下面分别从三个方面论述面向对象软件的特点及其对软件测试的影响。

1. 面向对象程序执行的动态性

面向对象软件与面向过程软件的一个主要区别在于,面向过程的程序鼓励过程的自治,但不鼓励过程之间的交互;面向对象的程序则不鼓励过程的自治,并且将过程(方法)

封装在类中，而类的对象的执行则主要体现在这些过程的交互上。

传统程序执行的路径是在程序开发时定义的，程序执行的过程是主动的，其程序流程可以用一个控制流图从头至尾地表示；而面向对象程序中方法的执行通常不是主动的，程序的执行路径也是在运行过程中动态地确定的。因此，对面向对象软件的测试应主要关注其动态模型。这也使得面向对象软件的集成测试不可能再沿用传统的集成策略，这一点将在面向对象软件的集成测试中进一步阐述。

2. 封装、继承和多态

面向对象程序设计语言提供了继承、封装、多态等机制，这些机制使开发出高质量软件成为可能，但同时也引入了新的测试视角，对面向对象软件需求分析的测试、面向对象设计的测试、面向对象软件的单元测试及集成测试都提出了新的要求，增加了面向对象软件测试的复杂性。

(1) 封装对测试的影响

类的重要特征之一是封装性，它把数据和操作数据的方法封装在一起，限制对象属性对外的可见性和外界对它的操作权限。封装简化了对对象的使用，为面向对象软件模块的安全性提供了强有力的保障，并减少了类之间的相互依赖性，提高了软件部件的可重用性，使得对软件的测试、维护等工作更易于进行。

但封装也给面向对象软件测试带来了新的问题，那就是如何获取对象的状态。什么是对象的状态？简单地说，对象的状态就是类中成员变量的值。

在面向对象系统中，系统的基本构造模块是封装了数据和方法的类和对象。每个对象有自己的生存周期，有自己的状态。对象的功能是在消息的触发下，由对象所属类中定义的方法与相关对象的合作共同完成。也就是说，面向对象软件的功能从宏观上看是各个对象相互作用的结果。

当对象执行某方法(成员函数)后，即在某种前置状态下执行某函数，对象的状态就会发生变化，即变迁到后置状态。值得注意的是，对象的状态不仅与执行的方法有关，还与对象的前置状态有关。测试时应关注对象状态的变迁，对每一对象设定前置状态，并判断经消息触发后的后置状态是否符合预期结果。即在输入数据部分给成员变量设定初始值，在输出部分判断成员变量的结果值是否符合预期结果。

由于类具有封装性，有时不能直接通过消息接口获取对象的状态，这时测试人员必须增添完成此种功能的函数。

例如，在一个堆栈类 Stack 中，其成员变量 h 代表了栈顶的高度。当堆栈不满时，每执行 1 次 push(x)，h 加 1；当堆栈不空时，每执行 1 次 pop()，h 减 1。但 h 是私有成员，对外界不可见，如何能够了解到程序执行后 h 的值是否正确地得到了改变呢？可以在 Stack 类中添加一个成员函数 return h()，用于返回 h 的值，这样便能观察到程序的执行结果。但这种方法增加了测试的工作量，且在一定程度上破坏了类的封装性。

(2) 继承对测试的影响

在面向对象程序中，继承机制使子类不但可以继承父类的特征，而且允许对继承的特征进行重定义。继承机制使得类的层次结构更加分明，继承是提高软件可重用性和可维护性的重要途径。但继承也给面向对象的软件测试带来了新的研究课题。

Weyuker 曾经提出了 11 条基于程序的测试数据集的充分性公理，Perry 与 Kaiser 根据 Weyuker 公理进行了讨论，得出与继承机制有关的结论如下：

① 在子类中重定义了某一个继承的方法，即使两个函数完成了相同的功能，重定义的方法在新的语境中仍需重新测试。

② 在多继承的情况下，从两个不同的父类所定义的同名(同型构)的特征中按不同的优先级(或选择方法)在子类中仅选择保留一个版本时，即使新得到的子类的结构与父类相同，仍然需要不同的测试数据集。

③ 若对父类中某一方法进行了重定义，仅对该方法自身或其所在类进行重新测试是不够的，还必须重新测试其他相关的类，例如子类和引用类。

④ 彻底复用的继承机制对每一个新的使用语境仍需要新的测试。也就是说，若一个类中的方法被其子类继承(未重定义)后，在子类的环境中的行为特征需要重新测试。

综上所述，继承并未简化测试问题，反而会使测试更加复杂。

(3) 多态对测试的影响

多态考虑的是类与类之间的层次关系，以及类自身内部特定成员函数之间的关系问题，是解决功能和行为的再抽象问题。多态是指类中具有相似功能的不同函数用同一个名称来实现，从而可以使用相同的调用方式来调用这些具有不同功能的同名函数。比如一个对象中有很多求两个数最大值的行为，虽然可以针对不同的数据类型，写很多不同名称的函数来实现，但它们的功能几乎完全相同，因此完全可以利用多态性，用统一的标识来完成这些功能，达到类的行为的再抽象。多态性为程序员编程提供了高度柔性、问题抽象和易于维护等特性，但它也给面向对象软件的测试带来了新的问题。

例如，设有类 A、类 B 和类 C 三个类，类 B 继承类 A，类 C 又继承类 B；成员函数 a()分别存在于这三个类中，但在每个类中的具体实现不同；同时，在程序中存在一个函数 fn()，该函数在形参中创建了一个类 A 的实例 Ca，并在函数中调用了方法 a()。程序运行时相当于执行了一个分情况语句 Switch，首先判定传递过来的实参类型(类 A 或类 B 或类 C)，然后再确定究竟执行哪一个类中的方法 a()。在测试时必须为每一个分支生成测试用例，以覆盖所有分支和所有程序代码。

从这个例子可以看出，多态性所带来的执行路径的不确定性，使得对传统的面向过程软件的静态分析方法不再适用。多态增加了测试用例选取的数量和难度。

3. 演化、迭代的开发模式

面向对象的开发往往用于大型软件项目，需求变更和方案变更较传统软件开发更为频繁。此外，基于面向对象方法中继承、封装等机制所提供的良好保障，使得面向对象软件的开发模式往往是演化、迭代的，因而不可能再用功能细化的观点检测面向对象分析和设计的结果。

综上所述，传统软件的测试方法和技术对面向对象软件已显得力不从心，针对面向对象软件的特点，应该有一套完整的测试方法和技术。

8.2.3 面向对象软件的测试模型

面向对象的开发模型可分为面向对象分析(OOA)、面向对象设计(OOD)和面向对象

编程(OOP)三个阶段。

OOA 和 OOD 很难截然区分开来,从 OOA 过渡到 OOD 是一个渐增式过程,它们之间的微小差别可这样理解：OOA 是一个分类的过程,它从问题陈述中直接反映问题域和系统行为的对象、类和类之间的联系孤立出来;而 OOD 则进一步说明为实现需求必须引入的其他类和对象,还应考虑对类结构进行改进,以提高软件开发的质量和效率。此外,OOD 还应提供刻画对象之间关系的表示方法。

至于 OOP,则是 OOA 和 OOD 的结果代码化,以最终实现要求的功能。

针对这种开发模型,结合传统的测试步骤的划分,可将面向对象软件的测试划分如下：

- 面向对象分析的测试。
- 面向对象设计的测试。
- 面向对象编程的测试。
- 面向对象软件的单元测试。
- 面向对象软件的集成测试。
- 面向对象软件的确认和系统测试。

其中,面向对象分析的测试和面向对象设计的测试是对分析和设计活动的测试,主要针对分析和设计产生的文档进行,是软件开发前期的关键性测试。面向对象编程的测试主要针对代码进行测试,该测试往往在面向对象单元测试和面向对象集成测试中实现。

面向对象软件的单元测试、集成测试和系统测试的层次是怎样划分的？与传统软件测试的划分有何不同？一般对面向对象软件的单元测试、集成测试和系统测试进行如下界定：

(1) 面向对象软件的单元测试即类级测试,考查封装在一个类中的方法和数据之间的相互作用。值得注意的是,面向对象软件的单元测试与传统的单元测试有很大的不同,传统单元测试是对单个功能模块(如函数)进行测试;而在面向对象系统中,基本模块——类不仅包含若干方法,还包含数据。对象中的数据和方法是一个有机的整体,方法的作用在于改变对象的状态,而对象的状态取决于类中成员变量的值,因而不能脱离类中的数据单独对方法进行测试。

(2) 面向对象软件的集成测试可以理解为类簇级测试,考查一组协同操作的类之间的相互作用。

(3) 面向对象软件的系统测试与传统的系统测试含义基本相同,考查由所有类和主程序构成的整个系统能否满足用户的功能性需求和非功能性需求。

从以上论述可以看出,由于面向对象软件与面向过程软件完全不同的思维方式和开发模式,使得面向对象软件的测试的视角从模块转变到了类。

由于面向对象的开发往往用到原型开发方法,这是一个演化式开发过程,因而软件测试的 H 模型对面向对象软件是适用的,应强调将测试活动作为一个独立的流程,可贯穿在软件开发生命周期的任何一个阶段,测试应是可迭代的。

面向对象软件的测试模型如图 8-1 所示。

下面将分别介绍面向对象分析的测试、面向对象设计的测试、面向对象编程的测试、面向对象软件的单元测试、面向对象软件的集成测试和面向对象软件的系统测试。

面向对象软件的系统测试
面向对象软件的集成测试
面向对象软件的单元测试
面向对象分析的测试
面向对象设计的测试
面向对象编程的测试
面向对象分析
面向对象设计
面向对象编程

图 8-1　面向对象软件的测试模型

1. 面向对象分析的测试

面向过程的分析方法是一个功能分解的过程，将一个软件系统看成可以分解的功能的集合。这种传统的功能分解分析法的着眼点在于一个系统需要什么样的信息处理方法和过程，以过程的抽象对待系统的需求。

而 OOA 不仅对过程进行抽象，更重要的是对数据进行抽象，它将问题空间中的实例抽象为对象，用属性和操作表示实例的特性和行为，用对象的结构反映问题空间的复杂实例及实例之间的复杂关系。对一个软件系统而言，与传统分析方法产生的结果相反，行为是相对稳定的，结构是相对不稳定的，这更充分反映了现实的特性，也使得面向对象软件对需求变更的适应性更强。

实际上，OOA 就是一个对类进行划分并建立类结构的过程，OOA 的成果为后阶段类的进一步选定和实现，类层次结构的优化和实现提供了平台。因此，OOA 对问题空间抽象的不完整，将会导致无法全面实现软件的功能，导致软件开发后期大量的修补工作；而一些冗余的对象或结构会带来不必要的设计和编码、测试负担。因而，对 OOA 的测试重点在其完整性和非冗余性。

尽管 OOA 的测试是一个不可分割的系统过程，但鉴于 Coad 方法提出的 OOA 的实现步骤，可将其划分为对认定的对象的测试、对认定的结构的测试、对认定的主题的测试、对定义的属性和实例关联的测试和对定义的服务和消息关联的测试 5 个方面。

(1) 对认定的对象的测试

OOA 中认定的对象是对问题空间中的结构，其他系统、设备，被记忆的事件，系统涉及的人员等实例的抽象。对它的测试可以从如下方面考虑：

① 认定的对象是否全面，是否问题空间中所有涉及的实例都反映在认定的抽象对象中。

② 认定的对象是否具有多个属性。只有一个属性的对象通常应看成其他对象的属性，而不被抽象为独立的对象。

③ 对认定为同一对象的实例是否有共同的、区别于其他实例的属性。

④ 对认定为同一对象的实例是否提供或需要相同的服务，如果服务随着不同的实例而变化，认定的对象就需要分解或利用继承性来分类表示。

⑤ 如果系统没有必要始终保持对象代表的实例的信息，提供或者得到关于它的服务，认定的对象则无必要。

⑥ 认定的对象名称应该尽量准确、适用。

(2) 对认定的结构的测试

在 Coad 方法中，认定的结构指的是多种对象的组织方式，用来反映问题空间中的复

杂实例和复杂关系。认定的结构分为两种：分类结构和组装结构。分类结构体现了问题空间中实例间一般与特殊的关系；组装结构（即聚集）则体现了问题空间的实例间整体与局部的关系。

对认定的分类结构的测试可从如下方面着手：

① 对于结构中的一种对象，尤其是处于高层的对象，是否在问题空间中含有不同于下一层对象的特殊可能性，即是否能派生出下一层对象。

② 对于结构中的一种对象，尤其是处于同一低层的对象，是否能抽象出在现实中有意义的更一般的上层对象。

③ 对所有认定的对象，是否能在问题空间内向上层抽象出在现实中有意义的对象。

④ 高层对象能否完全体现低层对象的共性。

⑤ 低层对象是否有高层对象共性基础上的特殊性。

对认定的组装结构的测试从如下方面入手：

① 整体（对象）与部件（对象）的组装关系是否符合现实的关系。

② 整体（对象）的部件（对象）是否在考虑的问题空间中有实际应用价值。

③ 整体（对象）中是否遗漏了反映在问题空间中有用的部件（对象）。

④ 部件（对象）能否在问题空间中组装新的有现实意义的整体（对象）。

(3) 对认定的主题的测试

主题是在对象和结构的基础上更高一层的抽象，是为了提供 OOA 结果的可见性而引入的，就如同文章中各部分内容的概要。对主题的测试应该考虑以下方面：

① 贯彻 George Miller 的“7＋2”原则，如果主题个数超过 7 个，就要求对属性和服务联系较紧密的主题进行归并。

② 主题所反映的一组对象和结构是否具有相同和相近的属性和服务。

③ 认定的主题是否是对象和结构更高层的抽象，是否便于理解 OOA 结果的概貌（尤其是对非技术人员的用户）。

④ 主题间的消息联系（抽象）是否代表了主题所反映的对象与结构之间的所有关联。

(4) 对定义的属性和实例关联的测试

属性是用来描述对象或结构所反映的实例的特性，实例关联则是反映实例集合间的映射关系。对属性和实例关联的测试应从如下方面考虑：

① 定义的属性是否对相应的对象和分类结构的每个现实实例都适用。

② 定义的属性在现实世界是否与这种实例关系密切。

③ 定义的属性在问题空间是否与这种实例关系密切。

④ 定义的属性是否能够不依赖于其他属性而被独立地理解。

⑤ 定义的属性在分类结构中的位置是否恰当，低层对象的共有属性是否在上层对象属性中体现。

⑥ 在问题空间中每个对象的属性是否定义完整。

⑦ 定义的实例关联是否符合现实。

⑧ 在问题空间中实例关联是否定义完整，特别需要注意一对多和多对多的实例关联。

（5）对定义的服务和消息关联的测试

定义的服务，就是定义的每一种对象和结构在问题空间所要求的行为。由于问题空间中实例间必要的通信，在 OOA 中需要定义相应的消息关联。对定义的服务和消息关联的测试应从如下方面进行：

① 对象和结构在问题空间的不同状态是否定义了相应的服务。

② 对象或结构所需要的服务是否都定义了相应的消息关联。

③ 定义的消息关联所指引的服务提供是否正确。

④ 沿着消息关联执行的线程是否合理，是否符合现实过程。

⑤ 定义的服务是否重复，是否定义了能够得到的服务。

2. 面向对象设计的测试

OOD 在 OOA 的基础上进一步引入了一些类，并对类结构进行了优化或进一步构造成类库，实现了对问题空间的进一步抽象。

OOD 与 OOA 的界限往往难以严格区分。OOD 是在 OOA 的基础上进一步优化和更高层的抽象。OOD 确定的类和类结构不仅是满足当前需求分析的要求，更重要的是通过重新组合或加以适当的补充，消除需求分析结果中的冗余部分，并能方便地实现功能的重用和扩展，以不断适应用户的需求。

对 OOD 的测试，可从如下三方面考虑：

① 对认定的类的测试。

② 对构造的类层次结构的测试。

③ 对类库的支持的测试。

（1）对认定的类的测试

OOD 认定的类可以是 OOA 中认定的对象，也可以是对象所需要的服务的抽象，以及对象所具有的属性的抽象。认定的类原则上应该尽量具有基础性，以便于重用和维护。可从如下方面测试认定的类：

① 是否包含了 OOA 中所有认定的对象。

② 是否能体现 OOA 中定义的属性。

③ 是否能实现 OOA 中定义的服务。

④ 是否对应着一个含义明确的数据抽象。

⑤ 是否尽可能少地依赖其他类。

⑥ 类中的服务是否为单用途的。

（2）对构造的类层次结构的测试

OOD 应对 OOA 阶段产生的类层次结构进行优化。在当前的问题空间，对类层次结构的主要要求是能在解空间构造实现全部功能的结构框架。为此，可从如下方面对构造的类层次结构进行测试：

① 类层次结构是否涵盖了所有定义的类。

② 是否能体现 OOA 中所定义的实例关联。

③ 是否能实现 OOA 中所定义的消息关联。

④ 子类是否具有父类没有的新特性。

⑤ 子类间的共同特性是否完全在父类中得以体现。

(3) 对类库支持的测试

对类库的支持虽然也属于类层次结构的组织问题，但它强调的是软件部件的重用性。由于它并不直接影响当前软件的开发和功能实现，因此，将它单独提取出来测试，也可作为对高质量类层次结构的评估。可从如下方面进行测试：

① 一组子类中关于某种含义相同或基本相同的服务，是否有相同的接口。

② 类中服务的功能是否较单纯，相应的代码行是否较少(建议不超过 30 行)。

③ 类的层次结构是否是深度大、宽度小的。

3. 面向对象编程的测试

面向对象编程的测试主要是通过面向对象软件的单元测试和面向对象软件的集成测试完成的，故在此不单独介绍。

4. 面向对象软件的单元测试

面向对象软件的单元测试即对面向对象软件中的基本模块——类进行测试。类是对若干方法和数据进行封装后形成的模块，如前所述，一个类对象有它自己的状态和依赖于状态的操作行为，该行为会使对象从现有状态变迁到其他状态。由于要关注对象的状态，面向对象软件的单元测试不能脱离类中的数据仅对方法进行测试，因而与传统软件的单元测试不同，面向对象软件的最小可测试单元不是单个方法，而是类或对象。

面向对象软件的类测试是由封装在类中的操作(方法)和类的状态行为所驱动的。类测试将对象与其状态结合起来，考查封装在一个类中的方法和数据之间的相互作用，对对象的状态行为进行测试。

类测试是面向对象软件测试发现错误的强有力的手段，类测试一般由编程人员完成。

类测试关注的内容主要如下：

① 对类中单个方法的测试。测试类中的单个方法，能否在消息的触发下实现正确的状态转移，即从现有状态变迁到预期的状态。

② 对类中方法间协作的测试在保证单个方法正确的基础上，还应该测试方法之间的协作关系，即测试类中的方法是否能够通过对象的状态正确地通信。

传统针对模块的测试用例设计技术，如逻辑覆盖、等价类划分、边界值分析和错误推测等方法，仍然可以视情况作为测试类中方法的技术，但由于类和传统的模块不一样的特点，仅仅依靠这些方法，是不能有效地完成类测试的。以下介绍目前作为应用或研究热点的一些典型的类测试方法。

(1) 基于状态的类测试

基于状态的测试把被测对象看作一个有限的状态机(FSM)，从该 FSM 导出测试用例。选择测试用例时可以利用基于 FSM 的测试所取得的研究成果。

基于状态的类测试方法的基本思想是，检查对象的状态在执行某个方法或方法序列后是否会转移到预期的状态；如果对象到达的状态不是预期的状态，则说明程序中含有错误。这里的“状态”有两种定义方式：设计状态和实现状态。它们分别对应于分析/设计

阶段和实现阶段所用的概念。实现状态是指在实现层次由对象的属性值确定的状态;设计状态是指建模阶段所认定的状态,处于较高的抽象层次,而不依赖于实现时的具体表示。设计状态在有些文献中称为逻辑状态或抽象状态。基于状态的测试方法最早由Turner 和 Robson 给出。该方法使用的是实现状态,通过跟踪监视对象数据成员的值的变化检验类中方法的执行是否正确。另一个有代表性的基于状态的测试方法是McGregor 给出的方法。该测试方法使用的是设计状态,一个“状态”被定义为属性值集合上的一个约束,这些约束互不相交。这种定义方式比 Turner 和 Robson 方法的状态定义更抽象,并且允许以子类型兼容的方式定义状态。

基于状态的测试可以充分测试类中的各种方法和可能的状态,符合类测试的特点,因此是当前类测试中用得较多、研究得也较多的技术之一,但其难点主要在于如何确定被测对象是否达到了正确的状态。

许多状态机测试方法(如 UIO 方法等),都是为每一个状态确定一个能够将它与其他状态相区别的迁移集或迁移序列,根据这些迁移上的消息及响应序列识别该状态,但这些方法往往较为复杂,并且经常难以奏效;另外,对于比较复杂的类,往往会产生状态爆炸的问题,问题的解决还有待进一步研究。

基于状态的测试可以很容易地推广到集成测试,只要能够为类簇(即具有协作关系的一组类)建立这样的状态模型。

(2) 基于 UML 的类测试

可用于面向对象软件单元测试即类测试阶段的 UML 图形主要有类图和状态图。

类图对图中类的属性和方法进行了列举说明,标识了类之间的关系,显然可作为类测试的重要依据。

状态图用来描述一个特定对象的所有可能状态及引起状态转移的事件,可表示单个对象在其生命周期中全部行为的事件顺序,因而它对类测试有着重要的意义。

基于状态图的类测试着眼于检测对象成员函数之间通过对象状态交互时产生的错误,即检查对象在现有状态下,执行某个方法后能否转移到预期的状态。使用该技术能够检验类中的方法是否能通过对象的状态正确地通信。

因为对象的状态是通过对象的数据成员值反映出来的,所以检查对象的状态实际上就是跟踪监视对象数据成员值的变化。

基于状态图的类测试,根据构造函数的规格说明设计测试用例,测试构造函数是否能正确设置对象的初始状态;还应着重检测对象对事件(消息)响应的正确性,即对于对象所有可能接受的事件,根据对象状态图设计测试用例,测试处于某一状态的对象对特定事件是否能正确地响应,并转移到预期的状态。

状态图展现了一个状态机,所以很多已有的对于状态机测试的有效方法都能很容易地被移植到基于状态图的测试中去。

(3) 基于数据流的类测试

基于数据流的类测试由传统的数据流测试发展而来。传统数据流测试的基本思想是,一个变量的定义,通过辗转的使用和定义,可以影响到另一个变量的值,或者影响到路径的选择等,因此可以选择一定的测试数据,使程序按照一定的变量的定义——使用路径

执行，并检查执行结果是否与预期的相符，从而发现代码的错误。这种测试思想也适用于面向对象的软件。但在类级和类簇级测试中，由于方法执行的先后顺序是动态决定的，因此必须首先得到类或类簇中正确的方法序列，测试用例则围绕这些方法序列中的类成员变量的定义——使用对产生。为一个类或类簇生成的测试用例集应能覆盖该类或类簇中所有类成员变量的定义——使用对。该方法主要着眼于类或类簇中的数据流，一般对其褒贬不一，有人认为这种测试是必不可少的，也有人认为这种方法破坏了类的封装性，不值得提倡。

（4）基于规范的类测试

基于规范的面向对象的测试技术，是指以需求和功能规范为基础的测试，通过分析软件的需求和功能规范选择和产生测试数据，重点测试一个作用于被测类的对象的消息序列是否将该对象置于“正确”的状态。Doong 和 Frankl 提出的 ASTOOT 方法是一种非常有代表性的基于形式规范的测试方法。ASTOOT 方法基于 ADT 的代数规范，使测试结果的判断归结为布尔值的比较，简化了测试先知问题，同时测试过程的自动化程度显著提高。对 ASTOOT 方法在此不详细介绍，读者可参考相关资料。

由于形式规范具有的严格、精确并支持机械推导的特点，形式规范与测试的结合是面向对象测试技术的重要发展方向。

（5）基于方法序列的类测试

面向对象程序中方法的调用是有一定顺序的，如果违反了这个顺序就会产生错误。Kirani 提出了一种方法序列规范（Method Sequence Specification，MtSS），方法序列规范 MtSS 规定了类或类簇中方法的执行顺序，如哪些方法必须按先后次序执行，哪些方法可以并发执行等。MtSS 能直接支持测试用例的生成，从 MtSS 产生的方法序列（或消息序列）可作为一个测试用例用于类测试或集成测试，检验这些类或类簇中的方法是否能够正确地交互。

由于该方法没有能够考虑类的状态，因此采用它进行的测试是不完全的。这种方法常常与其他测试方法结合使用。

5．面向对象软件的集成测试

面向对象的集成测试主要检测那些类相互作用时才会产生的错误。基于单元测试对成员函数行为正确性的保证，集成测试只关注系统的结构和内部的相互作用。

面向对象软件的集成测试可以分两步进行，先进行静态测试，再进行动态测试。

静态测试主要针对程序的结构进行，检测程序结构是否符合设计要求。现在流行的一些测试软件都能提供一种称为“可逆性工程”的功能，即通过原程序得到类关系图和函数功能调用关系图，例如 Rational 公司的 Rose C++ Analyzer 等，将可逆性工程得到的结果与 OOD 的结果相比较，检测程序结构和实现上是否存在缺陷，即检测 OOP 是否符合预期的设计要求。

面向对象软件集成测试的动态测试是软件测试的一大难点和研究热点，原因在于它与传统软件集成测试策略存在很大的不同。

面向对象程序的执行实际上是执行一个由消息连接起来的方法序列，而这个方法序列往往是由外部事件驱动的。即面向对象软件没有层次控制结构，针对传统软件的自底

向上或自顶向下的集成测试策略并不适用于面向对象软件。此外，在各个方法测试之后，每次选择一个方法集成到类中逐步进行测试通常也是不可行的，这是因为各个方法之间往往存在错综复杂的交互。因此，需要研究专门针对面向对象软件的集成测试策略（指动态的集成测试，以下同）。

集成测试有时也被称为类簇测试。所谓类簇（Cluster），是一组具有协作关系的类。集成测试（类簇测试）主要根据系统中相关类的层次关系，考查一组协同操作的类之间相互作用的正确性。即检查各相关类之间消息连接的合法性，继承机制的正确性，动态绑定执行的正确性，类簇协同完成系统功能的正确性等。

面向对象软件的集成测试方法主要如下。

(1) 基于 UML 的集成测试。统一建模语言 UML 中的若干图形也可以作为面向对象集成测试的用例设计依据。在此主要介绍基于类图和顺序图的集成测试。

① 基于类图的集成测试。类图说明了各对象类本身的组成，即类的属性和操作，描述了类之间继承、聚集、关联等静态关系。

根据类图中所描述的父类、子类的层次结构，将存在继承关系的类作为一个整体实施类簇测试，检查子类的继承特性与父类的一致性和动态绑定的正确性。其自然的测试顺序是父类在先，子类在后。父类可看作子类的公共部分，在父类经充分测试的前提下，子类测试应重点关注子类的独有部分以及父类和子类间的交互。

② 基于聚集关系的集成测试。聚集即反映类之间整体与部分的关系，可进一步划分成共享聚集（Shared Aggregation）和组成（Composition）。共享聚集的含义是部分可属于多个整体，如学生类和社团类之间的关系；组成的含义是部分只能依附于一个整体，若整体消失了，则部分也不复存在，例如某窗口的显示区域与该窗口的关系。

基于聚集关系的集成测试，其自然的测试顺序是部分类在先，整体类在后；在部分类经充分测试的基础上，测试整体类是否能按设计要求实现聚合。

另外，还可利用类图中的关联关系生成检验类之间关系是否正确实现了测试用例。

③ 基于顺序图的集成测试。顺序图描述对象之间动态的交互关系，着重体现对象间消息传递的时间顺序，因此它可以作为面向对象软件集成测试的依据。顺序图可以转换为流程图，这种流程图表示了对象间消息传递的顺序，与程序流程图在形式上极为类似，也包括了顺序、分支和循环等。采用基本路径法可导出流程图的基本路径集，路径集中的每一条路径都是一个消息序列，即测试用例。

(2) 基于数据流的集成测试

前面介绍过的基于数据流的类测试策略同样很容易扩展到面向对象软件的集成测试中来。

(3) 基于方法序列的集成测试

基于方法序列的集成测试即基于方法序列规范 MtSS 的集成测试。前面已介绍过的基于方法序列的类测试策略同样可用于面向对象软件的集成测试。

在使用以上介绍的某种集成测试方法时，应注意如下几点：

(1) 确定不需要重复测试的部分，从而精化测试用例，减少测试工作量。

（2）应注意使测试达到一定的覆盖标准。测试所要达到的覆盖标准可以是：对所有服务的一定覆盖率，依据类间传递的消息达到对所有执行线程的一定覆盖率，对所有状态的一定覆盖率等，同时也可以考虑使用现有的测试工具得到对程序代码的覆盖率。

（3）设计测试用例时，不但要设计满足预期功能的输入，还应该有意识地设计一些被禁止的例子，确认类能否恰当地处理这类输入，例如发送与类状态不相适应的消息，与要求不相适应的服务等。

（4）动态的集成测试有时也可以通过系统测试完成。

6. 面向对象软件的系统测试

为最后确认开发完毕的整个软件产品在实际运行环境下能否满足用户的全部需求，即能否实现需求规格说明书中指定的功能指标和满足性能、可靠性、安全性等非功能性指标，必须对软件产品进行严格、规范的系统测试；测试开发的软件作为最终系统的一个组成元素，与系统其他元素能否很好地协作运行，以满足用户需求。

与传统软件的系统测试相似，面向对象的系统测试不再考虑各模块之间相互连接的细节，集中检查用户可见的动作和可识别的输出，即验证软件系统确实实现了用户的需求。

系统测试的具体测试内容也与传统软件的系统测试内容相似，主要包括功能测试、压力测试和负载测试、强度测试、容量测试、安全性测试、健壮性测试、可靠性测试、配置测试和兼容性测试、用户界面测试、文档测试、安装测试、卸载测试等。

系统测试应该尽量搭建与用户实际使用环境相同的测试平台，应该保证被测系统的完整性，对没有的系统设备部件，也应有相应的模拟手段。

系统测试需要对被测的软件结合需求分析进行仔细的测试分析，建立测试用例。如以 UML 中的用例图为依据，对用例图进行逐层细化，以导出测试用例。从用例图导出测试用例的主要步骤如下：

① 标识出系统的功能。软件系统的功能是整个系统测试的基础，也是系统测试的出发点。如果这一阶段还不能准确描述，那么该系统的开发将很难取得成功，更不用说测试了。

② 建立高层用例图。用例开发从很高层次的视图开始。高层用例关注于待建系统的总的描述。

③ 建立基本用例图。高层用例只是对系统的总的描述，很少涉及细节。因此必须对其进行不断的精化。

④ 扩展基本用例图。如果该系统的功能相对较复杂，在基本用例中还没有完全描述清楚其功能，可以对基本用例进一步细化，进行扩展。

⑤ 根据基本用例的描述，导出系统测试用例，完成系统测试。当然，为了更好地完成系统测试工作，可以考虑在不同的覆盖范围内，对系统进行测试。

8.3 小结

本章介绍了与面向对象软件测试相关的内容。主要包括面向对象开发方法的概念、优点，面向对象软件测试的重要性，面向对象软件的特点对测试的影响，面向对象的软件

测试模型等。

在面向对象测试模型中介绍的面向对象单元测试和面向对象集成测试是本章最重要的部分。应重点掌握面向对象单元测试及集成测试的概念和策略。

习　　题

1. 结合面向对象开发的若干机制,试谈谈面向对象开发方法的优点。
2. 试谈谈面向对象软件的特点及其对软件测试的影响。
3. 面向对象软件的测试模型主要包括哪几种测试?它们的执行顺序如何?
4. 对面向对象分析的测试重点是什么?
5. 为什么说面向对象软件的最小可测试单元不是单个方法,而是类或对象?
6. 类测试关注的内容是什么?
7. 简述基于 UML 的类测试策略。
8. 简述面向对象软件的集成测试的作用。

第 9 章

软件测试自动化

本章要点：

- 自动化测试的优点。
- 对自动化测试的认识误区。
- 自动化测试的原理和方法。
- 主流测试工具。
- WinRunner 及其使用。

软件测试是一项繁重的任务，难以想象能完全通过手工操作来完成测试任务。自动化测试在软件测试中占有很大的比重。本章介绍软件测试自动化的若干基本概念，并对主流的测试工具和测试工具厂商进行了介绍，最后用较大篇幅介绍了如何用 WinRunner 进行功能测试。

9.1 自动化测试概述

1. 自动化测试的必要性

软件测试是一项需要付出极大时间和精力的工作，它贯穿软件开发的每一阶段。在一个软件项目的开发过程中，由于用户需求、开发方案等时常面临变更，软件测试常需迭代地执行，若不对测试过程和配置进行有效的管理，无法很好地完成既定的测试目标，发挥测试应具有的作用。

迭代的测试还意味着回归测试的执行。当回归的次数太多，而全部依靠人工来完成，任何人都会有难以忍受的感觉，从而使工作热情和工作质量大大降低。

事实上，软件测试本身的特点决定了测试中的很多工作（如回归测试中的大部分工作）是可以重复执行的，应将它们独立出来，交给测试工具来自动实现。

综上所述，基于测试过程管理的需求和软件测试活动的特点，用测试工具代替人工完成部分测试工作，也就是进行自动化测试，是有必要且可行的。

2. 自动化测试的优点

与人工测试相比，自动化测试（Automated Testing）主要具备如下优点：

(1) 可以提高软件测试的效率。使测试人员能更加专注于对软件中新的项目进行测试,从而提高对功能点及其他特性的测试覆盖率;有利于缩短软件的开发时间;使得测试人员在一定程度上从繁琐的人工测试中解脱出来,能有时间和精力去关注测试领域的前沿技术,提高自身的从业素质。

(2) 对测试配置(包括软件配置、硬件配置、人员配置等)进行有效的管理,并使测试配置在整个测试生命周期内得到复用,这在功能测试和回归测试中更有意义。

(3) 通过测试过程的自动化管理,使测试实施方可以通过流程的关键绩效指标(Key Performance Indicator,KPI)衡量测试过程的一致性和有效性,从而实现从软件质量保证向软件质量管理(Software Quality Management,SQM)的进化。

(4) 通过对测试过程进行规范的定义,避免过分依赖于个人,因而减少了出错的可能性,提高了测试的可靠性。

(5) 减少了人工测试,从而在一定程度上降低了测试成本。

(6) 可以执行一些手工不可能进行的测试。例如,通过测试工具模拟真实情况对软件进行压力测试、容量测试等。

(7) 使测试变得更加有趣。

软件测试自动化已成为测试行业及软件开发行业的大势所趋,软件测试自动化的水平在很大程度上代表了一个软件开发企业或测试外包提供商的技术实力。

9.2 自动化测试的引入和实施

9.2.1 对自动化测试的认识误区

虽然自动化测试有着许多好处,但也必须清醒地意识到,自动化测试的引入不是一蹴而就的,自动化测试本身也不是万能的。对于自动化测试,人们在认识上可能存在以下误区。

1. 自动化测试应完全取代人工测试

尽管自动化测试可以降低人工测试的工作量,但并不能完全取代人工测试。100%的自动化测试只是一个理想目标,即便一些如 SAP,Oracle ERP 等测试库规划得十分完善的套件,其测试自动化率也不超过 70%。这是因为自动化测试本身的特点决定了它发现软件缺陷的能力是有限的。一般说来,测试过程中 80%以上的缺陷是人工测试发现的,仅有不到 20%的缺陷是自动化测试发现的,而且这 20%要求企业具有较高的自动化测试实施水平。

所以,绝不要强行在测试的每个部分都采用自动化方式,一味地追求测试自动化只会给企业带来运作成本的急剧上升。

可以考虑使用自动化测试的情形主要包括:对主要功能的测试;容易自动化的测试;人工难以进行的测试;运行最频繁的测试和回报很快的测试。

2. 测试用例可完全由测试工具自动生成

有些测试工具可以实现部分测试用例的自动生成,例如 C/C ++ 单元测试工具 Visual

Unit 等，但主要用于检查未处理特殊输入而形成的错误。基本的功能测试仍然需要人工设定测试用例，完成白盒覆盖时，测试用例设计器也需要人工干预。这是因为测试工具无法自动了解程序的功能，自动生成的测试用例具有很大的局限性。依靠自动测试用例，通常只能发现异常之类的极端错误，大多数一般错误都是无法发现的。因而，测试用例主要仍是通过人工设计的。

3. 测试工具可以在任何场合使用

测试工具都是针对解决特定问题而开发的，有其功能的局限性。而且作为软件，测试工具也存在软件兼容性等方面的问题。

4. 自动化后测试效率立刻提高

一个企业决定引入测试工具实现自动化测试是需要做大量前期准备工作的，这些工作绝不是在很短时间内就能够完成的。

必须认识到，要实现软件测试自动化，仅仅依靠功能强大的自动化测试工具及自动化管理平台是不够的，测试自动化的实施效果更依赖于软件测试自动化的实现过程，以及在这个过程中所体现的软件质量管理和软件测试最佳实践。如果测试过程本身是不合理的，引入自动化测试只会使软件项目更加混乱。

此外，企业内部通常存在许多不同种类的应用平台，应用开发技术也不尽相同，甚至在一个应用中可能就跨越了多种平台；或由于开发时期的不同，可能导致同一应用的不同版本之间也存在技术差异。因此，测试自动化的引入必然给当前企业部门与部门间的合作，以及现有的应用平台及开发工具带来一定的冲击。

因而，企业在决定引入自动化测试之前，应先进行技术、资金等多方面细致的可行性分析。如果付出大量的人力、物力和财力，而企业自身的技术、流程管理水平、员工素质与实施自动化测试要求的标准还有较大距离，并且在短期内无法弥补，则自动化测试的引入只不过是一个空架子，将使企业得不偿失。

进行可行性分析后，若决定引入自动化测试，企业还需要进行一系列工作如下，以便为自动化测试的顺利引入创造良好的内部环境。

(1) 对开发及测试流程进行调整与改进，使之尽量合理、规范。

(2) 对现有的应用平台进行必要的整合。

(3) 对人员组织结构进行调整。

(4) 对需求、设计、编码、维护及配置管理等其他方面的工作进行优化。

(5) 对相关员工进行培训，包括测试流程、缺陷管理、测试工具使用等。

5. 自动化测试顺利引入后就一劳永逸了

顺利引入自动化测试工具后，自动化测试能否达到应有的效果还取决于实施过程中的多方面因素。

(1) 自动化测试能提高测试效率，但不会创造性地发现测试方案里没有设计缺陷。因此，应当由有经验的测试人员对自动化测试方案进行系统、周密的设计。

(2) 像其他软件开发项目一样，自动化测试代码也需要跟踪和维护，因此应使用配置管理工具来对脚本进行统一管理和维护。此外，引入自动化测试后，还应考查其工作流程

是否合理、规范。总之,对自动化测试实施的监督和评估是十分重要的。

9.2.2 自动化测试的实施流程

下面简要介绍自动化测试的实施流程。

(1) 选择测试工具。应根据本企业自身的业务需求和平台技术选择合适的测试工具。显然,由于业务需求和应用平台的多样性,不可能通过一个测试工具解决所有问题。选择测试工具的依据如下:

① 选择测试工具时应充分考虑到工具的可集成性和平台兼容性,因为各种自动化测试通常需要被集成起来,统一管理。

② 在经费有限的情况,应优先考虑测试流程管理工具,其次考虑性能测试工具,再考虑功能测试工具。

③ 应考虑测试工具对测试自动化方案可扩展性的支持,以满足企业技术的不断提升和业务需求的不断扩展。

④ 在考虑产品性价比的同时,产品的支持服务和售后服务的完善性也是值得关注的。

⑤ 应尽量选择主流测试工具,以便于通过行业间交流甚至网络等方式获得更为广泛、便捷的支持。

(2) 对于特殊的业务需求,若不能通过购买现有工具的方式来实现,则应自行开发测试工具。自行开发测试工具可以在工具中补偿被测软件缺乏的可测试性。

(3) 在全面实施自动化测试之前,可以先进行小规模的试验,为后阶段更有效地实施测试打下基础。

(4) 如软件生命周期有需求分析阶段一样,在对一个项目实施自动化测试之前,也应进行自动化测试的需求分析,也就是确定自动化测试应完成的功能指标。

(5) 根据自动化测试的需求分析文档,设计自动化测试方案。设计方案的正确与否具体决定了测试脚本发现缺陷的能力,因此应特别注意设计方案的合理性、全面性,当然也应避免冗余。

(6) 接下来可以进行测试脚本的开发。

(7) 使用配置管理工具对脚本进行统一管理和维护,对自动化测试的实施进行监督和评估。

自动化测试的实施流程如图 9-1 所示。

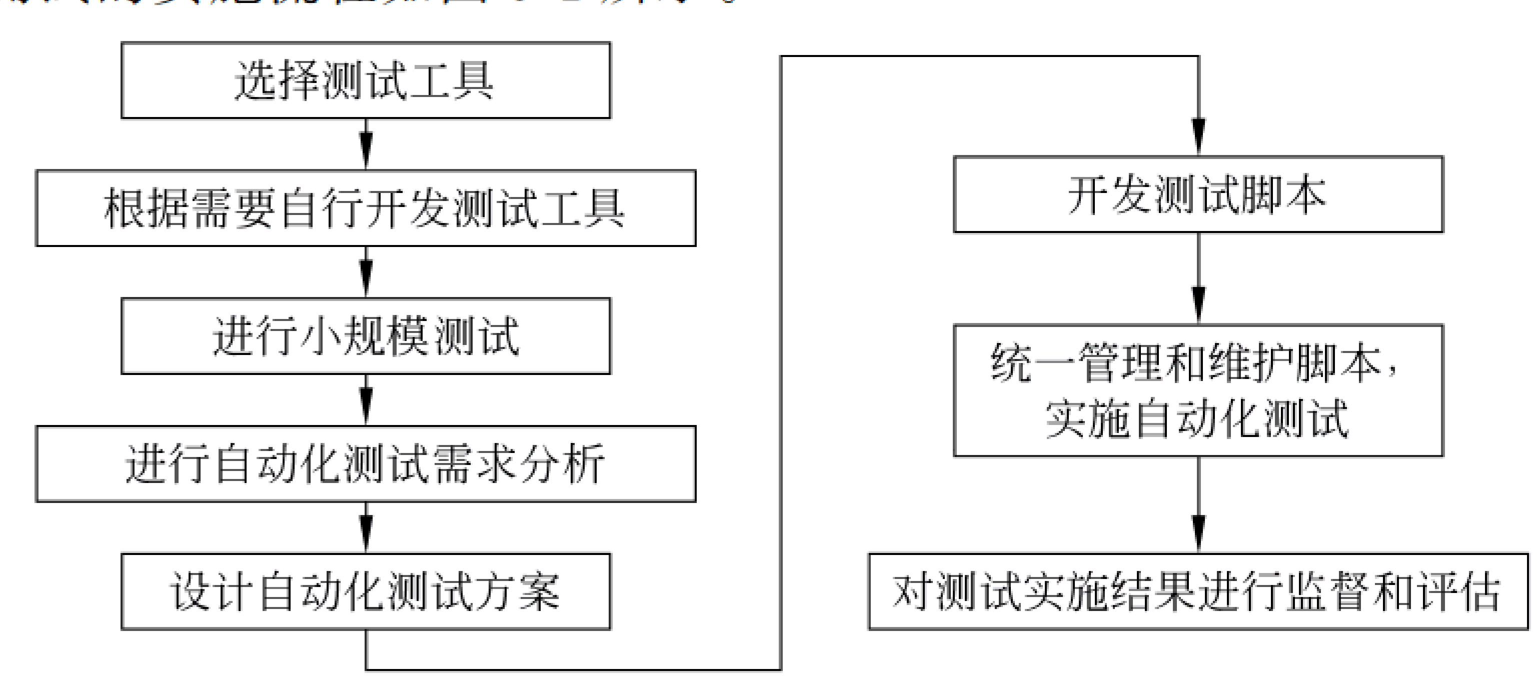


图 9-1 自动化测试实施流程

9.3　自动化测试的原理和方法

自动化测试基于的原理和方法主要有脚本预处理、脚本技术和自动比较技术，如图 9-2 所示。

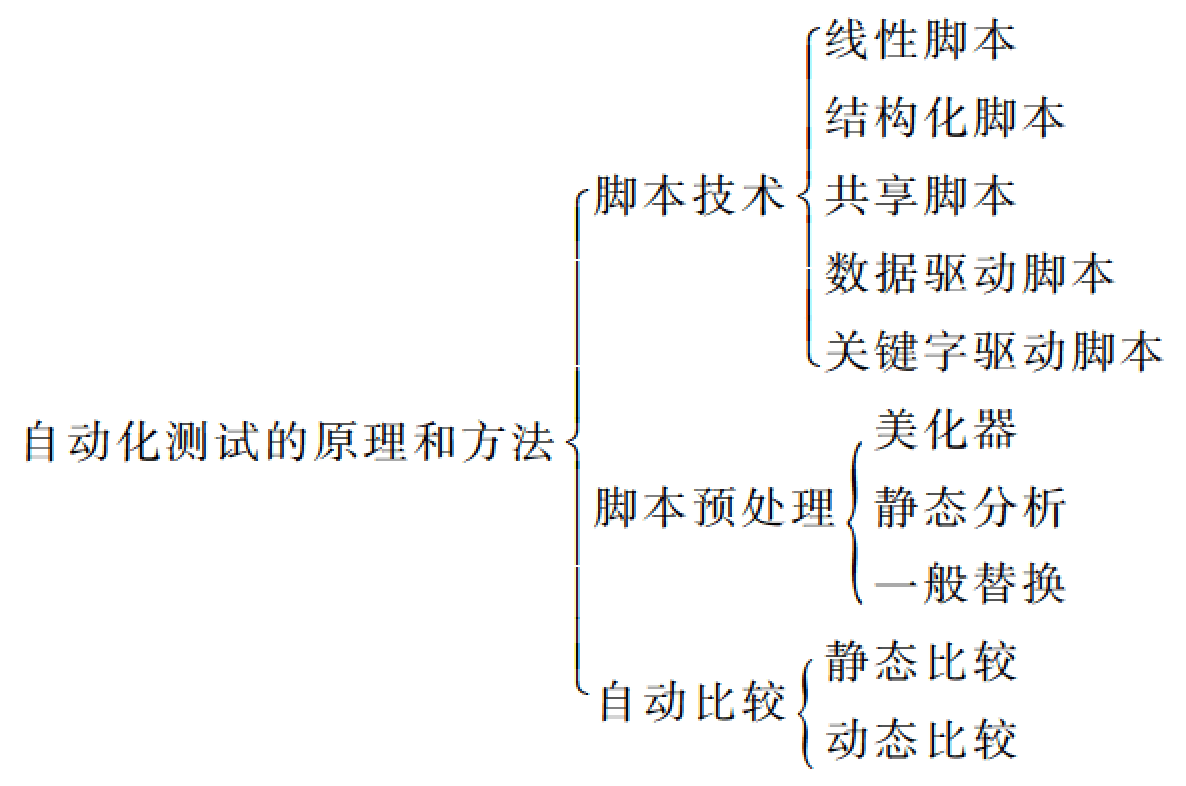


图 9-2　自动化测试的原理和方法

9.3.1　脚本技术

测试脚本（Test Script）是与特定测试对应的一系列指令（及数据），这些指令可以被测试工具自动执行。脚本是程序的一种形式。

脚本可通过录制测试的操作产生，再在其基础上进行修改；当然也可以直接用脚本语言编写脚本。

脚本中通常包含有如下信息：

- 同步（何时进行下一个输入）。
- 比较信息。
- 捕获何种屏幕数据，存储在何处。
- 从其他数据源读取数据信息。
- 控制信息。

脚本可分为线性脚本、结构化脚本、共享脚本、数据驱动脚本和关键字驱动脚本等。

1. 线性脚本

线性脚本是录制手工执行的测试用例得到的脚本。这种脚本包含所有用户的键盘和鼠标输入，所有录制的测试用例可以通过脚本完整地被回放。

在线性脚本中可以加入简单的指令，例如时间等待、比较等。

线性脚本适合于简单的测试（例如 Web 页面测试）、一次性测试、脚本的初始化、软件演示或培训等场合。

尽管线性脚本的录制方便，但其测试输入和比较是捆绑在脚本中的，无法实现脚本共享，且线性脚本修改代价大、维护成本高。

2. 结构化脚本

结构化脚本侧重于描述脚本中控制流程的结构化特性。结构化脚本中的控制流程或

为控制结构,或为调用结构。

控制结构中包括顺序、循环和分支结构。调用结构则是在一个脚本中调用另外脚本,当子脚本执行完成后再返回父脚本继续运行。

结构化脚本的优点是健壮性好、易于维护,且可以通过循环和调用减少工作量。

结构化脚本的主要缺点是测试数据仍然捆绑在脚本中,无法实现脚本共享。

3. 共享脚本

共享脚本是指脚本可以被多个测试使用,一个脚本可以被另外一个脚本调用。

共享脚本可以是在不同主机、不同系统之间共享脚本,也可以是在同一主机、同一系统之间共享脚本。这样可以节省生成脚本的时间,当重复任务发生变化时只需修改一个脚本。

共享脚本的优点如下:

- 可以较少的开销实现类似的测试。
- 维护开销低于线性脚本。
- 可以在脚本中增加更智能的功能。
- 通过共享脚本技术,还可以建立脚本库,达到最大限度的共享。

共享脚本的主要缺点是需要跟踪更多的脚本,给配置管理带来一定的困难。

4. 数据驱动脚本

数据驱动脚本技术将测试输入存储在独立的数据文件中,而不是绑定在脚本中。执行时是从数据文件而不是从脚本中读入数据。这种方法最大的好处是可以用同一个脚本做不同的测试。

使用数据驱动脚本,可以以较小的开销实现较多的测试用例,这可以通过为一个测试脚本指定不同的测试数据文件达到。将数据文件单独列出,选择合适的数据格式和形式,可将用户的注意力集中到数据的维护和测试上。

数据驱动脚本的优点如下:

- 可以快速增加类似的测试。
- 测试者增加新测试不需要掌握工具脚本语言的技术。
- 对第二个及以后类似的测试无额外的维护开销。

数据驱动脚本的缺点是初始建立的开销较大,且需要专业(编程)支持。

5. 关键字驱动脚本

关键字驱动脚本技术实际上是数据驱动脚本技术的逻辑扩展。它将数据文件变成测试用例的描述,用一系列关键字指定要执行的任务。在关键字驱动技术中,假设测试者具有某些被测系统的知识,因而不必告诉测试者如何进行详细的动作,只是说明测试用例做什么,而不是如何做。这样在脚本中使用的是说明性方法和描述性方法。描述性方法将被测软件的知识建立在测试自动化环境中,这种知识包含在支持脚本中。

例如,为完成在网页浏览时输入网址,一般的脚本需要说明在某某窗口的某某控件中输入什么字符。而在关键字驱动脚本中,可以直接是在地址栏中输入网址,甚至更简单,仅说明输入什么网址。

关键字驱动脚本的数量不随测试用例的数量变化，而仅随软件规模而增加。这种脚本还可以实现跨平台的用例共享，只需要更改支持脚本即可。

关键字驱动脚本技术常与数据驱动脚本技术一起使用。

9.3.2　脚本预处理

脚本的预处理是指脚本在被工具执行之前必须进行编译，预处理功能通常需要工具支持。脚本预处理的功能主要有美化器、静态分析和一般替换。

美化器是一种对脚本格式进行检查的工具，必要时可以对脚本进行转换，以符合编程规范的要求。美化器可以让脚本编写者更专注于技术性工作。

静态分析对脚本或表格执行更重要的检查功能，检查脚本中出现的和可能出现的缺陷。通常，该测试工具可以发现一些如拼写错误或不完整指令等脚本缺陷，类似于程序设计中的 PC-Lint 和 LogiScope 的功能。

一般替换也就是宏替换。可以让脚本更明确、易于维护。使用替换时，应注意不要执行不必要的替换。

9.3.3　自动比较技术

测试验证是检验软件是否产生了正确输出的过程，是通过在测试的实际输出与预期输出之间完成一次或多次比较实现的。比较器可以检测两组数据是否相同，功能较齐全的比较器还可以标识有差异的内容。但比较器并不能告诉用户测试是否通过或失败，需要用户自行判断。

自动比较的内容可以是多方面的，包括基于磁盘输出的比较，例如对数据文件的比较；基于界面输出的比较，例如对显示位图的比较；基于多媒体输出的比较，例如对声音的比较；还包括其他输出内容的比较。

自动比较可分为静态比较和动态比较。动态比较是在测试过程中进行比较，静态比较是不在测试过程中进行比较，而是将测试结果存入数据文件，再通过工具进行比较。

自动比较还可以分为简单比较和智能比较。简单比较查看实际输出与预期输出是否完全相同。智能比较则允许用已知的差异比较实际输出和预期输出。例如，要求比较包含日期信息的输出报表的内容，如果使用简单比较显然是不行的，因为每次生成报表的日期信息肯定是不同的。这时就需要智能比较，忽略日期的差别，比较其他内容如日期格式。智能比较需要使用到较为复杂的比较手段，包括正则表达式的搜索技术、屏蔽的搜索技术等。

9.4　对产品可测试性的考虑

软件产品一般会用到下面三种不同类别的接口：命令行接口（Command Line Interfaces，CLIs）、应用程序接口（API）和图形用户接口（GUI）。这些接口都是被测试的对象。

从本质上看，API 接口和命令行接口比 GUI 接口容易实现自动化测试。导致 GUI 自动化测试复杂的原因主要有：

- 需要手工完成部分脚本；
- 把GUI自动化测试工具和被测试的产品有机地结合在一起需要面临技术上的挑战；
- GUI设计方案的变动将直接带来GUI自动化测试复杂度的提高，而GUI设计方案的变动又是时常发生的。

因此，为了降低自动化测试复杂度，应确定被测产品是否包含API接口或命令行接口。有些时候，这两类接口隐藏在产品的内部，需要细致地分析才能找到。

对于隐藏可编程接口，InstallShield这一当前非常流行的制作安装盘的工具就有命令行选项，采用这种选项可以实现非GUI方式的安装盘，从提前创建好的文件中读取安装选项。这种方式比采用GUI安装方式更简单也更可靠。

为了让API接口测试更为容易，应该把接口与某种解释程序，例如Tcl，Perl或者Python绑定在一起。这使交互式测试成为可能，并且可以缩短自动化测试的周期。采用API接口方式，还可以实现独立的产品模块的单元测试自动化。

如果确实没有找到命令行接口或者API接口，开发人员最好对产品进行改造，以提供命令行接口或者API接口，从而支持产品的可测试性。

9.5 测试工具概述

测试工具可以是测试管理工具、性能测试工具、功能测试工具或白盒测试工具，这些主要是HP Mercury，IBM Rational，Segue，Compuware和Empirix等公司的产品，以及相当数量的开源测试工具。在商业化测试工具中，HP Mercury公司的产品所占有的市场份额最大。

注意，所谓开源工具是指开放源代码的测试工具。与商业测试工具相比，开源测试工具具有小巧、灵活、易扩展、免费等特性。但开源工具缺乏使用培训和技术支持，工具的用户界面一般也较为粗糙。技术水平较高的测试人员，或经费紧张的中小企业，可考虑使用开源工具。

9.5.1 主流测试工具

1. 测试管理工具

测试管理工具对测试配置和测试过程进行管理，对缺陷进行跟踪管理。

主要的测试管理工具如下：

- HP Mercury公司的TestDirector（基于Web集成的全球测试管理工具）。
- IBM Rational公司的TestManager（管理所有测试活动和工件的核心平台）。ClearQuest（变更和缺陷跟踪、管理工具）；ClearCase（管理变更和资源，控制开发过程中发展演化的一切内容）和RequisitePro（需求管理工具）。
- Segue公司的SilkCentral Test Manager（测试管理工具）和SilkCentral Issue Manager（缺陷管理工具）。
- Empirix公司的e-Monitor（应用监控工具）。

- Compuware 公司的 QADirector（基于分布式应用的高级测试管理工具）和 TrackRecord（变更和缺陷跟踪、管理工具）。
- T-Plan 公司的 T-Plan（测试过程管理工具）。
- Atlassian 公司 JIRA（基于 J2EE 技术的缺陷管理工具）。
- Bugzilla（缺陷管理开源工具）。
- Bugzilla Test Runner（基于 Bugzilla 缺陷管理系统的测试用例管理开源工具）。
- CVS（版本控制开源工具）。
- TestLink（基于 Web 的测试管理和执行开源工具）。
- Mantis（基于 Web 的缺陷管理开源工具）。

2. 功能测试工具

典型的功能测试工具如下：

- HP Mercury 公司的 WinRunner（功能测试及回归测试工具）和 QuickTest Pro（B/S 系统的功能测试工具）。
- IBM Rational 公司的 Robot（客户机/服务器应用程序的功能和性能测试工具），XDE Tester（Java 和 Web 应用的功能测试和回归测试工具）和 TeamTest（用于功能测试、性能测试、回归测试、测试管理，降低软件发布风险，缩短软件上市时间）。
- Compuware 公司的 QARun（功能测试和回归测试工具）。
- Segue 公司的 SilkTest（功能测试和回归测试工具）。
- Empirix 公司的 e-Tester（Web 应用的功能测试工具）。
- Radview 公司的 WebFT（模拟单用户对 Web 系统进行功能测试）。
- WebInject（针对 Web 应用程序和服务的功能测试开源工具）。
- MaxQ（Web 功能测试开源工具）。

3. 性能测试工具

典型的性能测试工具如下：

- HP Mercury 公司的 LoadRunner（预测系统行为和性能的负载测试工具）。
- IBM Rational 公司的 Robot 和 TeamTest。
- Compuware 公司的 QALoad（企业级负载测试工具），EcoTools（高层次的性能监测工具）和 EcoScope（应用性能优化工具）。
- TestBytes（数据库测试数据自动生成工具）。
- Segue 公司的 SilkPerformer（企业级性能测试工具）。
- Empirix 公司的 e-Load（Web 应用的压力测试工具）。
- Radview 公司的 WebLoad（Web 压力测试工具）。
- MicroSoft 公司的 Web Application Stress Tool（WAS）（Web 压力测试工具）。
- Parasoft 公司的 Insure ++（针对 C/C ++ 语言的实时性能监控及分析优化工具）和 Jcontract（针对 Java 的实时性能监控及分析优化工具）。
- DBMonster（SQL 数据库的压力测试开源工具）。

- OpenSTA(B/S结构的性能测试开源工具,基于CORBA体系)。
- Apache JMeter(针对HTTP或FTP服务器应用程序的性能测试开源工具,基于Java)。
- Web Application Load Simulator(LoadSim,网络应用程序的负载模拟器,开源工具)。

4. 白盒测试工具

典型的白盒测试工具如下:

- Compuware公司的NuMega DevPartner Studio(白盒测试工具)。
- IBM Rational公司的PurifyPlus(单元测试和可靠性测试工具,包括PureCoverage,Purify和Quantify)。
- Parasoft公司的Jtest(Java单元测试工具),Parasoft公司的Jcontract(Java实时性能监控及分析优化工具);C++ Test(C/C++单元测试工具);CodeWizard(C/C++代码静态分析工具);.test(.Net代码分析和动态测试工具)和Insure++(C/C++实时性能监控和分析优化工具)。
- 凯乐软件公司的Visual Unit(国产的C/C++单元测试工具,申请了多项专利,拥有一批创新的技术)。
- XUnit系列开源框架。这是目前最流行的单元测试开源框架,根据支持的语言环境不同,可分为JUnit(Java),CppUnit(C++),DUnit(Delphi),PhpUnit(PHP),AUnit(Ada),DotUnit(.NET),HttpUnit(Web),HtmlUnit(Web),JsUnit(Javascript),PhpUnit(PHP),PerlUnit(Perl),XmlUnit(XML)等。

9.5.2 IBM Rational软件自动化测试解决方案

IBM Rational软件自动化测试解决方案的三个最佳成功经验是尽早测试、连续测试和自动化测试,并在此基础上提供了完整的软件测试流程和一整套软件自动化测试工具,为企业自动化测试提供了优良的解决方案。

1. IBM Rational软件自动化测试解决方案的最佳成功经验

(1) 尽早测试

IBM Rational公司主要在以下三个方面为其产品用户提供尽早测试的软件工程技术。

首先,软件的整个测试生命周期是与软件的开发生命周期基本平齐的过程。即当需求分析基本明确后,就应该基于需求分析的结果和整个项目计划进行软件的测试计划;伴随着分析设计过程,同时应该完成测试用例设计;当软件的第一个发布出来后,测试人员应立即基于它进行测试脚本的实现,并基于测试计划中的测试目的执行测试用例,对测试结果做评估报告,从而通过各种测试指标实时监控项目质量状况,提高对整个项目的控制和管理能力。

其次,应进行迭代测试。即将原来的整个软件开发生命周期分成多个迭代周期,在每个迭代周期都进行测试,这样在很大程度上提前了软件系统测试发生的时间,以尽早发现关键问题,降低项目风险和项目开发成本。

第三，IBM Rational 公司还扩展了传统软件测试从单元测试、集成测试到系统测试、验收测试的阶段划分，将整个软件的测试按阶段划分成开发人员测试和系统测试两个阶段。通过将测试时间提前，尽早地发现软件中的缺陷，降低软件测试成本。

(2) 连续测试

在开发过程中的每次迭代开始前都要根据项目当前的状态和所要达到的阶段性目标制定迭代计划，而且每个迭代中都包括需求、设计、编码、集成、测试等一系列开发活动，都会增量式地集成一些新的系统功能。通过每次迭代，都将产生一个可运行的系统，通过对这个可运行系统的测试来评估该次迭代是否达到预定的迭代目标，并以此为依据制定下一次迭代的目标。

由此可见，在迭代式软件开发的每个迭代周期，都会进行软件测试活动，整个软件测试的完成是通过每个迭代周期不断地增量测试和回归测试实现的。这就是连续测试的概念。

连续测试不但能够持续地提高软件质量，同时也使系统测试的尽早实现成为可能，从而有效地控制开发风险、减低测试成本并保证项目进度。

(3) 自动化测试

IBM Rational 公司通过一套完整的软件测试工具，在实现测试管理流程的基础上，同时涵盖了功能测试、性能测试和可靠性测试的自动化测试需求，且通过工具之间的集成完成测试资源的整合。

2. IBM Rational 软件测试流程

IBM Rational 软件的测试流程，不仅包含完整的软件测试流程框架，还提供了内嵌软件测试流程测试管理工具的支持。

(1) IBM Rational 软件测试流程框架

IBM Rational 统一开发流程(Rational Unified Process，RUP)提供了一套完整的测试流程框架，软件测试团队可以它为基础，根据业务发展的实际要求，定制符合团队使用的软件测试流程。RUP 中的软件测试流程如下：

① 制定测试计划。

② 设计测试。

③ 实施测试。

④ 执行测试且评估测试。

⑤ 若还需进行回归测试，转步骤②，否则结束整个测试流程。

(2) 利用 IBM Rational 软件测试管理平台实现软件自动化测试流程

IBM Rational 在 RUP 测试方法论的基础上构建了软件自动化测试管理平台工具 TestManager，通过与测试需求管理工具 RequisitePro、缺陷管理工具 ClearQuest 的完美集成，实现了对整个软件测试生命周期的管理，可以帮助软件测试团队快速建立软件测试平台和测试管理流程。

TestManager 能与 Rational 其他工具无缝地集成，整合了从测试需求、测试计划、测试设计、测试实施、测试执行到测试结果分析、测试报告的自动生成等整个测试生命周期的管理活动，完成对包括产品的功能性、可靠性和性能等全方位的测试，方便测试管理人员进行软件测试过程监控和有关软件质量的各种量化指标的采集、分析；同时，统一组织

各种测试用例及测试脚本等测试配置，高效地进行回归测试。

3. IBM Rational 软件自动化测试工具的功能

除了测试流程管理外，IBM Rational 软件自动化测试工具还可以完成如下功能。

(1) 实现自动化功能和性能测试

IBM Rational 测试方案通过 TestManager 和 Robot，在实现测试管理流程的同时，能够完成功能测试和性能测试。

① 自动化功能测试。功能测试主要围绕 Windows 图形界面、字符终端和 Browser 界面进行测试。客户端可以使用 VC，VB，PB，Delphi 等编制的软件、各种字符终端软件或者浏览器应用，通过自动录制形成测试脚本，实现自动化功能测试和回归测试。

IBM Rational 的功能测试解决方案的目标，是使功能性测试变得更简单、有效并可重复执行，从而提升功能测试能力。它主要具有以下特点：

- 对各种环境(IDE)中开发的应用程序、字符终端软件，便捷地完成包括测试计划、测试设计、测试实施、测试执行和测试结果分析等全部测试流程。
- 便于录制或编写各种功能测试脚本，实现自动化的功能测试和回归测试。
- 利用数据池方便地解决大批量数据驱动的功能测试。
- 便捷地完成分布式功能测试，可以一次测试多种测试平台。
- 能够自动完成功能测试需求覆盖，确保应用程序满足产品规格说明和测试计划的每一个业务需求。

IBM Rational Robot 是一个强大的功能测试工具，它可以对使用各种集成开发环境(IDE)和语言建立的软件应用程序，创建、修改并执行自动化的功能测试、分布式功能测试、回归测试和集成测试。

Robot 是测试.NET 应用程序的首选工具，因为它是唯一可以为.NET 控件(包括 VB .NET，C＃，J＃和 Managed C ++)的测试提供全面的本机支持的测试工具。Robot 可以将基于 Microsoft Visual Studio .NET WinForms 和 WebForms 构架的应用程序的功能测试、分布式功能测试和回归测试自动化，并将.NET 应用程序的配置测试加以简化和自动化。

IBM Rational Robot 是 IBM Rational Suite® 产品家族中的一员。Rational Suite 家族提供了综合的开发平台，可统一开发团队、优化个体效率，并简化 IBM Rational 解决方案的实施。

Robot 的主要作用及特点如下：

- 支持多种 IDE。Microsoft VisualStudio .NET，Oracle Developer/2000，Delphi，PeopleSoft 和 PowerBuilder。
- 支持多种语言。Java，HTML，DHTML，Visual Basic，Visual C ++ ，ActiveX 和 XML。
- 自动 GUI 功能测试。
- 执行分布式功能测试。
- 测试所有.NET 本机控件，包括 VB .NET，C＃，J＃和 Managed C ++ 。
- 在 SQA Basic，VB 和 VU(Virtual User，虚拟用户)环境下创建并编辑脚本。

Robot 编辑器提供有色代码命令，并且在强大的集成脚本开发阶段提供键盘帮助。

- 脚本回放阶段收集应用程序诊断信息，Robot 与 Rational Purify，Quantify，Pure Coverage 集成，可以通过诊断工具回放脚本，在日志中查看结果。
- 执行完整的性能测试。Robot 和 TestManager 协作可以记录和回放脚本，这些脚本有助于断定多客户系统在不同负载情况下是否能够按照用户定义的标准运行。

为了提高对 Java 和 Web 开发的应用软件功能测试的支持，IBM Rational 的功能测试的解决方案还提供了 IBM Rational XDE Tester。它主要用于在 Windows 和 Linux 平台上基于 Java 和 Web 开发的应用软件的功能测试，是业内最先进、省时的 Java 和 Web 应用的测试工具，不仅节省了 Java 和 Web 测试人员的时间，而且使应用的测试更加彻底。XDE Tester 尤其适用于使用 IBM WebSphere Studio，Eclipse 和 Rational XDE Developer 等开发平台进行软件开发的团队。

XDE Tester 最重要的特性如下：

- 测试 Java 应用程序(J2EE，J2SE，SWT，AWT/JFC 控件)。
- 测试 Web 应用程序(HTML，DHTML，XML，JavaScript，Java Applet)。
- 用 Java 编写测试脚本，方便了测试的自定义用户界面和 Java 控制，提高了项目的效率。
- 使用 ScriptAssure 技术，帮助创建灵活、可重用的测试脚本，极大地提高了脚本的可维护性，且可以对动态数据进行无缝验证。
- 与 Rational 团队沟通工具集成。
- 与 Eclipse Shell，WebSphere Studio 和 Rational XDE Developer 无缝地集成，使测试平台扩展嵌入 Eclipse Shell，WebSphere Studio 和 Rational XDE Developer 开发平台，统一了测试和开发环境。

只有 IBM Rational XDE Tester 才能为测试人员提供：

- 将 Java 作为测试脚本语言。
- ScriptAssure 技术。
- 强大的测试开发环境 Eclipse Shell。
- 在 IBM WebSphere Studio 和 IBM Rational XDE Developer 内安装测试工具。

IBM Rational XDE Tester 集成了一组强大的功能，包括测试管理、缺陷跟踪、测试脚本版本控制和需求管理，统称为 IBM Rational 团队统一平台(IBM Rational Team Unifying Platform)。平台上的工具为团队内的协调提供了极大的便利条件，加速了开发的进程。

ScriptAssure 是一组独特的功能，具备以下优点：

- 允许设置测试脚本对 GUI 变更的敏感性，无须在应用的各个版本间更新测试脚本。
- 允许在修改界面之前就开始测试 GUI 功能。
- 允许检查一系列可接受的响应，从而方便地验证动态应用内容。ScriptAssure 技术可以节省测试人员的实际测试时间。

IBM Rational XDE Tester 利用 Eclipse 或 IBM WebSphere Studio 的强大功能作为其测试开发环境，这为测试人员提供了下列优势：

- 专业的开发环境。
- 具备代码的现代编辑器，协助创建和编辑测试脚本。
- 完整的调试器，可以非常容易地识别和解决测试脚本中的问题。
- 一条直接与开发人员沟通的途径。

② 自动化压力测试。Rational 性能测试解决方案可以方便灵活地模拟各种负载模型，完成以查找响应时间瓶颈、系统吞吐量、最大并发虚拟用户等为目标的各种要求的性能测试。具体包括如下：

- 利用 TestStudio 可以完成对压力测试的测试需求、测试计划、测试设计、测试实施、测试执行和测试结果分析等整个测试生命周期的管理。
- 利用 TestStudio 中的 Test Suite，能够方便地完成压力测试对负载模型的各种要求，包括：建立复杂的 Scenario 模型；准确模拟复杂负载的时序控制；基于 Transaction 的负载分析；建立面向目标的事务负载模型；响应时间精确到 1/100 秒；支持不同虚拟用户的不同 IP 地址模拟和准确的波特率模拟。
- 利用 TestStudio，能够方便地完成压力测试过程中各种指标的观测。
- 利用 TestStudio，能够方便地完成压力测试结果分析和各种结果报告的生成。

(2) 实现自动化的可靠性测试和单元测试

IBM Rational 软件测试工具 PurifyPlus 主要用于单元测试和可靠性测试。PurifyPlus 包含 Rational Purify，Quantify 和 PureCoverage 三个产品。

Rational Purify 主要针对软件开发过程中难以发现的内存错误和运行时的错误。它能在软件的开发过程中自动地发现错误，准确地定位错误，提供完备的错误信息，从而减少了调试时间，帮助开发团队找出缺陷并最终形成高质量的产品。

Rational Quantify 主要解决软件开发过程中的性能问题。它能在软件开发过程中方便地查明并显示应用程序的性能瓶颈，从而确保整个应用程序的质量和性能。Rational Quantify 给开发团队提供了一个性能数据的全局图形化视图，使他们从开发流程的开始时就注重性能问题。

Rational PureCoverage 提供应用程序的测试覆盖率信息。在软件开发过程中，它能自动找出代码中未经测试的代码，保证代码测试覆盖率；帮助开发人员确保开发质量，并使质量保证人员能够评价测试工作的效果；还可针对每次测试生成全面的覆盖率报告，可以归并程序多次运行所生成的覆盖数据，自动比较测试结果以评估测试进度。

在回归测试中，Rational Purify，Quantify 和 PureCoverage 通常与 Robot 结合使用，以完成特定的功能。

(3) 实现实时系统的自动化测试

IBM Rational Test RealTime 主要适合于开发实时系统和具有较高要求的非实时系统的软件开发，可以帮助测试团队快速建立单元测试、集成测试、系统测试等测试能力。它提供的自动测试(包括单元测试、集成测试、系统测试)、代码覆盖、内存泄漏检查、性能分析及 UML 跟踪等重要特性，帮助软件测试团队在系统崩溃前发现并修复软件缺陷。

其主要功能特性如下：

- 通过源代码分析，自动生成在目标上运行所需的测试脚本和测试程序。除了利用测试脚本指定测试数据外，不需要手工编码。而且在测试报告中，测试结果与源代码相联，简化了代码修改。
- 通过代码自动插桩进行代码覆盖率、内存泄漏及性能瓶颈进行分析，并与测试用例建立关联。
- 通过把测试结果和观察结果与被测代码关联，把测试作为开发的一个重要部分，实现开发、测试、评估的同步。
- 利用通用的、低开销而且易于移植的目标适配技术(Target Deployment Port，TDP)，使得测试与编译器、连接器、调试器及目标结构无关，从而实现了跨多开发环境、多目标结构。
- 利用 UML Trace 功能观察应用运行状态，并通过状态机模型覆盖实现测试用例与模型的关联，充分利用了模型和代码级测试的长处。
- 与 ClearCase，ClearQuest 和 RUP 集成，在同一集成环境中完成对测试文件进行版本控制，提交和修改变更请求。

9.5.3 HP Mercury 软件自动化测试解决方案

HP Mercury 公司是全球应用实施领域的领导者，为企业自动化测试提供了一整套解决方案。

HP Mercury 的应用实施产品率先提供了以业务为中心的生命周期方法，用于优化预生产期间的质量和性能。测试和开发人员可以制定更加明智的“投入使用”决策，减少软件缺陷，节省部署新软件或软件升级所耗费的时间和成本，同时确保应用程序随时都能提供预期的业务功能。

Mercury Quality Center 提供在各种 IT 及应用程序环境下的自动化软件测试和质量保证。而 Mercury Performance Center 率先提供了生命周期方法，可以优化应用程序性能，帮助确保应用程序能够扩展支持一定数量的用户、事务处理量和性能水平。

对 Mercury Quality Center 和 Mercury Performance Center 中最重要的测试工具可以进行如下分类：测试管理工具 TestDirector，性能测试工具 LoadRunner，功能测试工具 QuickTest Professional 和 WinRunner。

1. 测试管理工具 TestDirector

TestDirector 是业界第一个基于 Web 的测试管理系统，它使得在公司内部或外部进行全球范围内的测试管理成为可能。

TestDirector 通过单一浏览器操作界面登录和使用，并支持组织机构间的协同作业。它向测试人员、开发人员及其他相关人员提供了一个包含所有测试信息的中央数据仓库和访问数据仓库时的统一界面，使各类人员在不同的地方就能交互测试信息、执行测试活动，消除了组织机构间、地域间的障碍。

TestDirector 可以集成 Mercury 测试工具(WinRunner，QuickTest Professional，QuickTest Professional for MySAP. com Windows Client，LoadRunner 和 Visual

API-XP)以及第三方和自定义测试工具、需求和配置管理工具,可以无缝地与所选择的测试工具通信,提供一种完整的解决方案,使应用程序测试完全自动化。

TestDirector 将测试过程流水化,通过一个整体的应用系统集成了测试管理的各个部分,包括需求管理、测试计划、测试安排及执行、缺陷管理等核心模块。

使用 TestDirector,项目组中的各类人员就可以围绕统一的测试流程各司其职,例如业务分析人员定义应用需求和测试目标;测试经理和项目主管制定测试计划,并开发测试案例;测试自动化工程师创建自动化的脚本,并将脚本保存于存储器;SQA 测试人员运行手动测试和自动测试,汇报执行结果,并输入缺陷;开发人员登录数据库中检查并修复缺陷;项目经理创建应用状态报告,并管理资源的分配情况;产品经理对应用发布的就绪状况做出决策。

TestDirector 使得在软件发布和项目审核过程中节省大量的时间。它可以避免项目组中处于不同位置的各类人员的重复劳动,测试数据损失和沟通不良等问题。另外,TestDirector 还可以帮助 QA 团队制定长期目标、短期目标,以及确定应用程序是否准备就绪的标准。

下面简要介绍 TestDirector 的功能模块。

(1) 需求管理

TestDirector 的 Web 界面简化了需求管理的过程,提供了一个较直观的机制将测试计划、测试案例和应用功能需求联系起来。需求管理还能很好地处理需求变更的情况,能及时根据需求的变更调整测试计划。

TestDirector 的需求管理保证了整个测试流程的统一,确保了上线应用能够满足用户的最终需要。

(2) 计划测试

TestDirector 的 Test Plan Manager 通过直观的结构帮助测试人员将应用需求转化为具体的测试计划。

Test Plan Manager 提供了多种方式建立完整的测试计划,例如:

- 直接建立计划。
- 根据用 Requirements Manager 所定义的应用需求,通过 Test Plan Wizard 快捷地生成测试计划。
- 若已经将计划信息储存于 Microsoft Word 或 Excel 中,可将其导入到 Test Plan Manager。

Test Plan Manager 还能进一步地帮助测试人员完善测试设计和以文件形式描述每一个测试步骤。它还能为每一项测试建立附属文件,例如 Word,Excel,HTML,以更详尽地描述测试计划。

TestDirector 还能简化将人工测试切换到自动测试脚本的过程。

建立的所有测试计划都存放在一个中央存储器中,可通过可折叠式目录树可以方便地进行查看,且使测试小组可以便捷地重复使用测试计划和独立的测试案例,用于未来的应用发布。

（3）安排和执行测试

Test Lab Manager 允许既定测试在无人操作的情况下不间断地运行，或者在系统处于最低资源需求的状态下运行，且可将所有测试结果保存在中央存储器中。通过定义不同测试间的依赖关系，测试人员可以逼真地模拟真实的业务流程，同时更方便地维护和重复使用测试案例。

（4）缺陷管理

TestDirector 的 Defect Manager 支持整个缺陷生命周期——从初始问题发现，直至缺陷修复和验证修复，这确保了缺陷在定位之前不会被忽视。在任何新的缺陷被提交之前，TestDirector 会检查数据库以发现相似的缺陷，最大限度地减少重复缺陷，并消除了手动检查的需要。

（5）图形化和报表输出

TestDirector 的图形化和报表输出功能，能够在测试的任一环节帮助测试人员对数据信息进行分析，能以标准的 HTML 或 Word 形式快速地生成正式测试报告，测试分析数据还可简便地输入到一种工业标准化的报告工具，如 Excel，ReportSmith，CrystalReports 及其他类型的第三方工具。测试报告有利于项目组对应用状态做出决策。

2. 功能测试工具 WinRunner

WinRunner 是企业级功能测试工具。通过自动录制、检测和回放用户等应用操作，WinRunner 能够有效地帮助测试人员对复杂的企业级应用的不同发布版本进行测试，确保跨平台的、复杂的企业级应用能够成功地发布。

WinRunner 的最大特点是能快速、批量地完成针对功能点的测试，因此十分有利于回归测试。

此外，WinRunner 支持程序风格的测试脚本，高水平的测试人员可通过对脚本编程建立流程复杂、功能强大的测试，且能实现对测试脚本的重用。

针对大多数编程语言和 Windows 技术，WinRunner 提供了较好的集成、支持环境，因而适合于基于 Windows 平台的应用程序的功能测试。

对于 WinRunner，将在 9.6 节中更详细地介绍。

3. 功能测试工具 QuickTest Professional

QuickTest Professional（QTP）是基于 Web 的企业级自动化功能测试工具。它为每一个重要软件应用和环境提供功能和回归测试自动化的行业最佳解决方案。

与 WinRunner 相比，QTP 侧重于测试 B/S 架构，而 WinRunner 侧重于测试 C/S 架构的软件。

下面介绍 QuickTest Professional 的工作流程。

（1）创建测试

用 QuickTest Professional 创立一个测试，只需记录下一个标准的业务流程，例如下一张订单或建立一个新的商家账户。QuickTest Professional 直观的记录流程能让任何人在 GUI 上通过点击鼠标就可建立完整的测试；也可以编辑测试指令来建立复杂的测试。QuickTest Professional 将两种测试创建方式结合在一个环境中，用户可根据企业自

身情况选用。

(2) 插入检查点

在记录一个测试的过程中,可在需要检查应用程序反应的地方插入检查点。通过检查点来检查 GUI 对象、位图及数据库。在这个过程中,QuickTest Professional 捕捉数据,并作为期望结果储存下来,供以后进行验证。

(3) 检验数据

除了创立并运行测试,QuickTest Professional 还能验证数据库的数值,从而确保交易的准确性。例如,在测试创建时,可设定哪些数据库表格和记录资料需要检测。在重放时,测试程序就会核对数据库内的实际数值与预想的数值。QuickTest Professional 能自动显示检测结果,在有修改、删除或插入的记录上用突出标识以引起注意。

(4) 增强测试

与 WinRunner 相似,QuickTest Professional 的数据驱动向导(Data Driver Wizard)使其用户能通过简单地使用鼠标将一个记录下的业务流程转化为一个数据驱动测试。

针对相当数量的企业应用中的非标准对象,QuickTest Professional 的虚拟对象向导(Virtual Object Wizard)能识别以前未知的对象,不必特别编写代码,从而提高测试脚本的可读性和测试质量。

(5) 运行测试

创建测试并插入检查点、添加一定的功能后,就可执行测试。QuickTest Professional 执行测试时,它会自动操作应用程序,正如一个真实用户根据记录流程执行每一步的操作。而且,它的意外处理功能为测试排除干扰,包括消息和警报。

(6) 分析结果

测试运行后应对测试结果进行分析。QuickTest Professional 的互动式报告工具能够提供详尽的、易读的报告,列出在测试中发现的差错和出错的位置,帮助对所得的结果进行分析。点击按钮,还能进一步获取任何未被包括在此测试范围内的错误的详尽资料。

(7) 维护测试

每当应用程序被更改后,就应对其进行测试。有了 QuickTest Professional,则不必应用程序每改动一次,就建立新的测试。QuickTest Professional 会帮助测试人员创立在程序应用周期内可重复使用的测试,因而极大地节省了测试时间和资源。

4. 负载测试工具 LoadRunner

LoadRunner 是业界领先的预测系统行为和性能的负载测试工具。它通过模拟成千上万名实际用户和员工的行为并进行实时性能监测,对企业的应用系统进行测试,发现并精确定位整个企业架构中存在的问题。通过使用 LoadRunner,企业能最大限度地缩短测试周期,优化系统性能,并加速应用系统的部署时间。此外,LoadRunner 支持最广泛的协议标准和技术,可以为企业特定的应用环境量身订制解决方案。

LoadRunner 的主要功能和特征如下。

(1) 轻松创建虚拟用户

使用 LoadRunner 的 Virtual User Generator,能方便地创建系统负载。该引擎能够生成 agent 或虚拟用户模拟业务流程和真正用户的操作行为,方法是先记录业务流程(如

下订单或机票预定)，然后将其转化为测试脚本。利用虚拟用户，可在安装了 Windows，UNIX 或 Linux 系统的机器上同时产生成千上万个用户访问量，因而极大减少了负载测试所需硬件和人力资源。

另外，利用 LoadRunner 的 TurboLoad 专利技术可以获得最高适应性水平，因为 TurboLoad 能够帮助测试人员建立每天与数十万在线用户和数以百万计的点击数相匹配的负荷量。

用 Virtual User Generator 建立测试脚本后，可通过 Data Wizard 自动实现其测试数据的参数化。Data Wizard 直接连接数据库服务器，从中获取所需数据并直接输入到测试脚本。从而利用不同的实际发生数据测试应用程序，反映出系统的负载能力。

利用 LoadRunner 也可以控制某些行为特性，以进一步确认能够模拟真实用户，例如控制交易的数量、交易频率、用户的思考时间和连接速度等。

(2) 创建切实可行的负载方案

建立起虚拟用户之后，就需要制定切实可行的负载方案，即确定在每一个负载服务器上运行的业务流程和数量的实际用户。用 LoadRunner 的 Controller，能方便地组织起多用户的测试方案。Controller 的 Rendezvous 功能提供了一个互动的环境，在其中既能建立起持续且循环的负载，又能管理和驱动负载测试方案。

利用其日程计划服务可以将测试过程自动化。此外，LoadRunner 通过其 AutoLoad 技术还提供了更多的测试灵活性。使用 AutoLoad，可以根据目前的用户人数事先设定测试目标，优化测试流程。

(3) 定位性能问题

LoadRunner 内含集成的实时监测器，使得在负载测试过程中，可以随时观察到应用系统的运行性能。这样，就可以在测试过程中从客户和服务器的双方面评估这些系统组件的运行性能，从而更快地发现问题。

此外，利用 LoadRunner 的 ContentCheck，可以判断负载下的应用程序功能是否正常。ContentCheck 在虚拟用户运行时，检测应用程序的网络内容，从中确定是否有错误内容传送出去。它的实时浏览器提供了一个从终端用户角度观察到的程序性能情况。

(4) 分析结果以精确定位问题

测试完毕，LoadRunner 收集汇总所有测试数据，并能提供高级的分析和报告工具，以便迅速查找到性能问题并精确定位。使用 LoadRunner 的 Web 交易细节监测器，可以了解将所有图像、框架和文本下载到每一网页上所需要的时间。另外，Web 交易细节监测器分解用于客户端、网络和服务器上端到端的反应时间，便于确认问题，定位查找真正出错的组件。例如可将网络延时进行分解，以判断 DNS 解析时间，连接服务器或 SSL 认证所花费的时间。

(5) 重复测试保证系统发布的高性能

负载测试是一个重复过程。每次处理完一个出错情况，都需要对应用程序在相同的方案下再进行一次负载测试，以验证所做的修正是否改善了运行性能。

(6) Enterprise Java Beans 的测试

LoadRunner 完全支持 EJB 的负载测试。这些基于 Java 的组件运行在应用服务器

上，提供广泛的应用服务。通过测试这些组件，可以在应用程序开发的早期就确认并解决可能产生的问题。

利用 LoadRunner 可以方便地了解系统性能。它的 Controller 允许重复执行相同的测试方案；基于 HTML 的报告能提供一个比较性能结果所需的基准，以此衡量在一段时间内，有多大程度的改进并确保应用成功。

(7) 最大化投资回报

所有 HP Mercury 的产品和服务都是集成设计的，能完全相容地一起运作。由于测试脚本具有相同的核心技术，都是来自于 LoadRunner 和 Mercury 主动负载测试服务 ActiveTest 的，所以它们可以被重复用于性能监测。借助于 Mercury 的监测方案、Topaz 和 ActiveWatch，通过重复利用测试脚本，可以平衡投资收益，且能为测试前期部署和实时监测获取一个全面的应用性能管理解决方案。

(8) 支持无线应用协议

随着无线装置数量和种类的增多，测试计划需要同时满足传统的基于浏览器的用户和无线互联网设备，如手机和个人数字式助手。LoadRunner 支持两项最广泛使用的协议——无线应用协议(WAP)和商务模式(i-mode)。此外，对整个从入口到网络服务器架构进行负载测试，只需要通过记录一次脚本，就可以完成无线互联网系统的测试。

(9) 支持 Media Stream 应用

LoadRunner 还能支持流媒体(Media Stream)应用。为了保证终端用户得到良好的操作体验和高质量流动，需要对流媒体应用程序进行检测。使用 LoadRunner，可以记录和重放任何现代流行的多媒体数据流格式，以此诊断系统的性能问题，分析流应用的质量。

(10) 完整的企业应用环境的支持

LoadRunner 支持广泛的协议，可以测试各种 IT 基础架构。

9.6　使用 WinRunner 进行功能测试

9.6.1　WinRunner 简介

1. **启动** WinRunner

单击 WinRunner 8.0 的桌面图标(本节以 WinRunner 8.0 为例讲解 WinRunner 及其使用)，或在“程序”菜单中选中 WinRunner 8.0，可以启动 WinRunner，如图 9-3 所示。WinRunner 启动时的界面如图 9-4 所示。

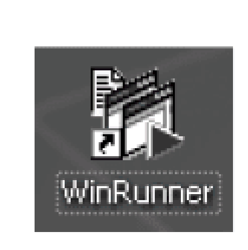




图 9-3　WinRunner 8.0 开始菜单



图 9-4　WinRunner 8.0 启动界面

然后就会出现 WinRunner Add-in Manager(插件管理器)对话框供用户加载插件,如图 9-5 所示。在插件管理器中,可选择支持 ActiveX Controls,PowerBuilder,Visual Basic 或 WebTest 插件,其他插件则需向 MI 公司购买。建议在此不要选择所有插件,因为这样可能会对录制或执行脚本造成不良影响。



图 9-5　WinRunner 的插件管理器

若不选中 Show on startup 复选框,则在启动时此界面不再出现。也可以在进入 WinRunner 后在 Tools 菜单 General Options 级联菜单中再选择 Startup 选项,在其中进行设置,以决定是否在启动 WinRunner 时显示插件管理器界面并显示时间。

2. WinRunner 主界面

启动 WinRunner 后,进入 WinRunner 主界面,在主界面 File 菜单中选择 Open 选项,即可打开一个已有的测试,或单击 New 选项新建一个测试。单击 New 选项新建一个测试后,将进入如图 9-6 所示主界面。

图 9-6　WinRunner 主界面

3. WinRunner 主界面各组成部分介绍

（1）标题栏

标题栏显示了目前正在编辑的测试。

（2）主菜单

菜单中包含了 WinRunner 的所有功能。

（3）工具栏

在 WinRunner 主界面中的工具栏如下：

- “文件”工具栏。通过工具栏中的工具，可以新建、打开、保存和打印测试文件，设置测试属性，查看测试结果，查看帮助。
- “调试”工具栏。通过工具栏中的工具可以调试测试脚本。
- “测试”工具栏。通过工具栏中的工具，可以选择运行模式，以不同方式录制脚本，运行脚本等。
- “用户”工具栏。工具栏中包含创建测试时常用的命令，用户可对其进行定制。

（4）工作窗口

工作窗口用于显示当前正在录制或编辑的脚本，如图 9-7 所示。

（5）状态栏

状态栏位于 WinRunner 主界面的底部，显示出当前的运行状态或所选择的命令。

4. WinRunner 测试模式

在 WinRunner 中，使用录制脚本的方式生成测试时，包括上下文相关模式（Context

```
# Flight Reservation
    set_window ("Flight Reservation", 4);
    menu_select_item ("File;Open Order...");

# Open Order
    set_window ("Open Order", 2);
    button_set ("Order No.", ON);
    edit_set ("Edit_1", "4");
    button_press ("OK");

# Flight Reservation
    set_window ("Flight Reservation", 3);
    menu_select_item ("File;Fax Order...");

# Fax Order No. 4
    set_window ("Fax Order No. 4", 1);
    edit_get_text("# Tickets:",tickets);
    edit_get_text("Ticket Price:",price);
    edit_get_text("Total:",totals);
    ################################################
    if(tickets*price==totals)
        tl_step("Verify totals", 0, "The totals field is OK.");
    else
```

Noname1　Lesson_1

图 9-7　WinRunner 工作窗口

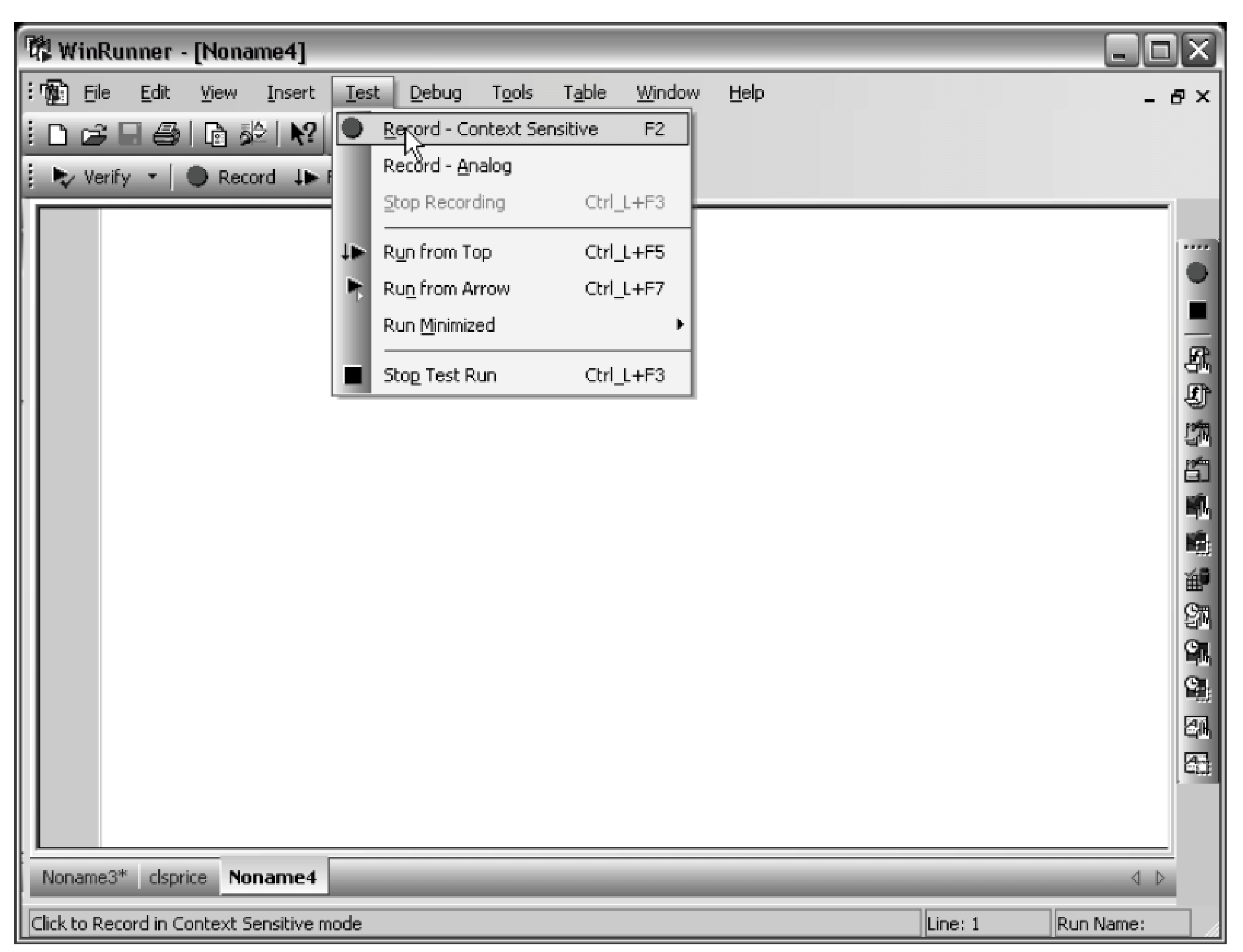


图 9-8　使用上下文相关模式

Sensitive mode)和模拟模式(Analog mode)两种录制测试的模式,如图 9-8 所示。

(1) 上下文相关模式

在该模式下,WinRunner 能够识别每个用户点击的 GUI 对象(例如窗口、按钮或列表)和执行的操作(如拖放、单击或选择),并忽略这些对象在屏幕上的物理位置。用户对被测软件进行的每一次操作,都会被 WinRunner 记录下来,保存在测试脚本中。脚本语言描述了用户选取的对象和操作动作。当用户录制脚本时,WinRunner 会对用户选取的每个对象进行唯一描述,并写入 GUI map(映射)文件中。在 WinRunner 中,GUI map 文件和测试脚本被分开保存并维护。这样,当软件界面发生变化时,只需更新 GUI map 文

件即可。所以，基于上下文相关的测试脚本非常容易实现重用。执行测试只需要运行测试脚本即可，通过运行测试脚本，WinRunner 可以模拟用户的操作步骤来运行被测软件。WinRunner 会自动从 GUI map 中读取对象描述，并在被测软件中查找符合这些描述的对象，这些对象在窗口位置中的变化不会影响 WinRunner 的查找。由此可见，上下文相关模式是以记录动作为目标的录制模式。

（2）模拟模式

这种模式是以记录鼠标轨迹和键盘操作为目标的录制模式。在该模式下，WinRunner 能够记录鼠标点击、键盘输入和鼠标在二维平面上的精确运动轨迹。这种模式对于那些需要追踪鼠标运动的测试非常有用，如绘图软件等。

9.6.2　WinRunner 测试流程

WinRunner 的测试流程大致包括 6 个步骤：创建 GUI map 文件、创建测试脚本、调试脚本、运行测试、检查测试结果和提交缺陷。

1. 创建 GUI map 文件

在 WinRunner 中，可以通过 GUI map 文件识别被测试应用程序中的 GUI 对象。一般可以使用两种方式创建 GUI map 文件，一种是使用 RapidTest Script Wizard 来自动学习对象，用它产生 GUI map 文件；另一种是在录制脚本时，通过用户单击对象，由 WinRunner 捕捉对象的描述，并将其加入 GUI map 文件中。

启动 RapidTest Script Wizard 的方法如图 9-9 所示。如果选择了第二种方式创建 GUI map 文件，实际上可以直接进入到步骤 2（创建测试脚本）。由此可见，在使用 WinRunner 进行测试时，首先要确定 GUI map 文件的创建方式。

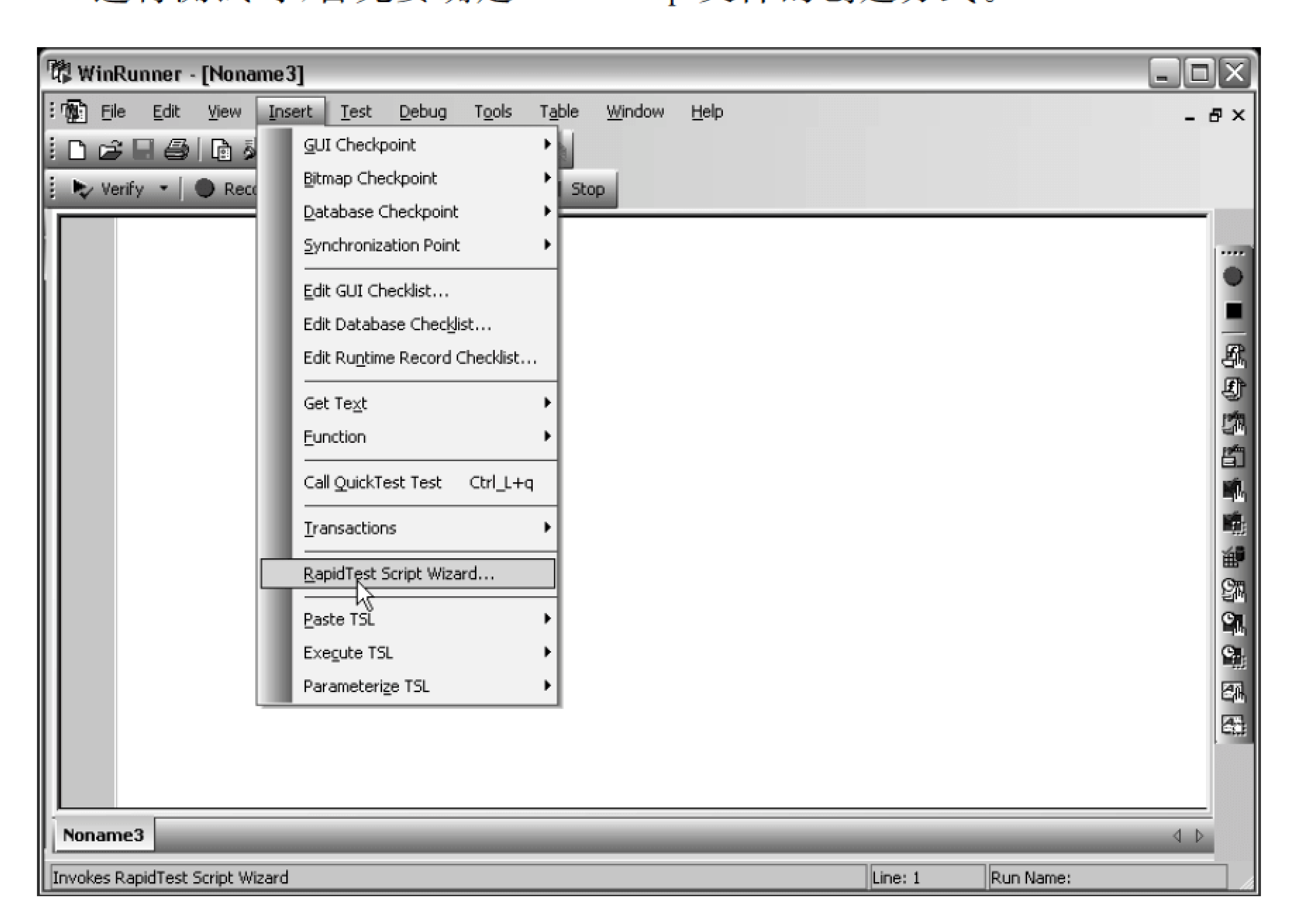


图 9-9　启动 RapidTest Script Wizard

需要注意的是，如果在 WinRunner 的 General Options 对话框中的 GUI map file mode 选项区中选中 GUI map file per test 单选按钮（单击 Tools 菜单的 General Options 选项），如图 9-10 所示；或者选择 WebTest 插件（单击 File 菜单的 Test Properties 选项），如图 9-11 所示，那么在图 9-9 中，RapidTest Script Wizard 菜单项就不会出现。

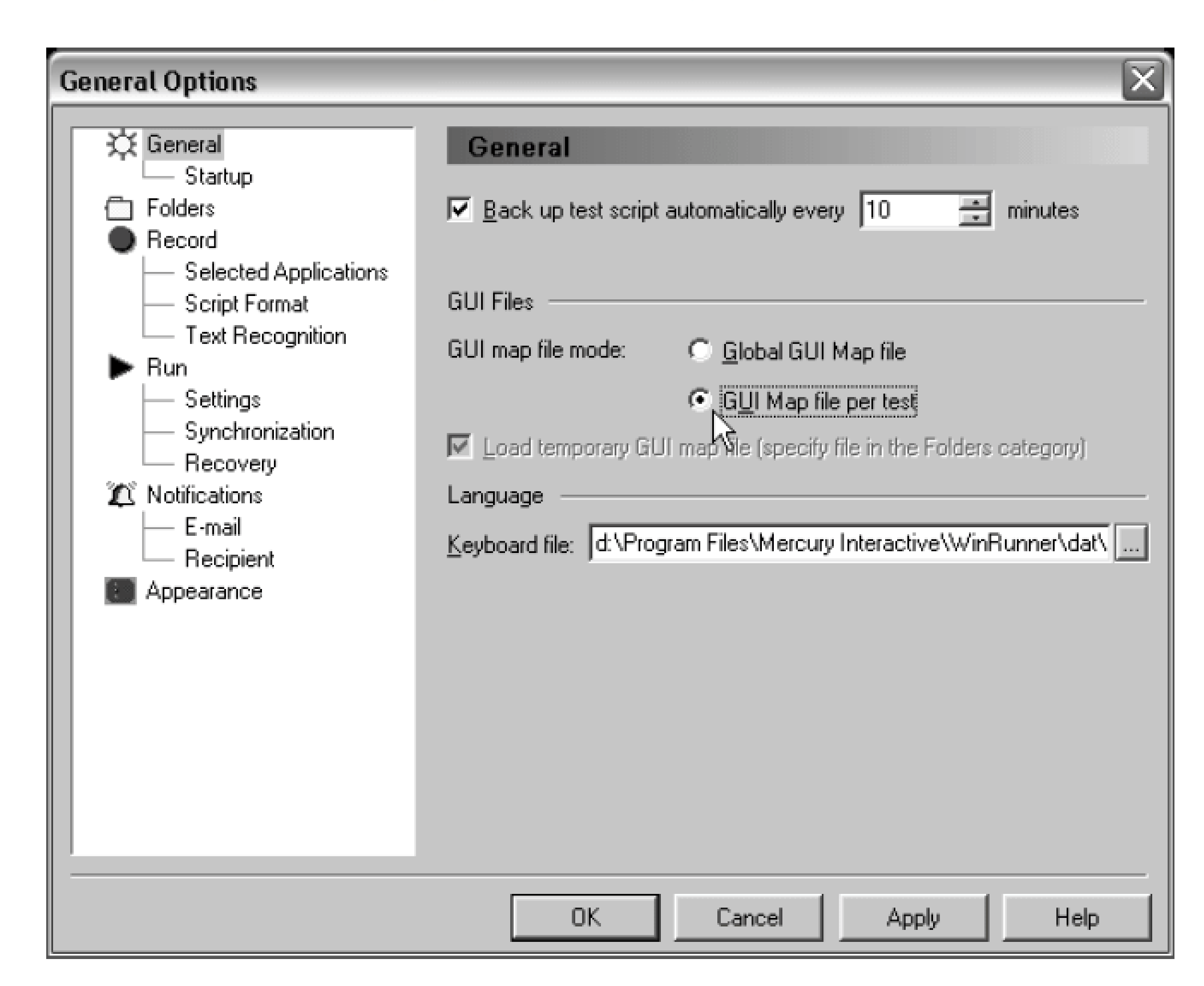


图 9-10　GUI map file mode 设置



图 9-11　插件设置窗口

2. 创建测试脚本

测试脚本的创建有录制和编程两种方式，或者两者的组合。在录制测试脚本时，可以在用户想要检查被测试应用程序响应的地方插入检查点（Checkpoint），以确定程序是否能够正确地运行。这些检查点包括 GUI 对象检查点、图像检查点、数据库检查点和文本

检查点。

3. 调试脚本

在运行测试脚本前，将运行模式设置为调试运行模式(Debug Run mode)，可以对测试脚本进行调试，如图 9-12 所示。通过调试，可以比较精确地查找到出现错误的位置。在实际测试过程中，可以使用 WinRunner 提供的 Step，Step into，Step out 功能调试脚本。

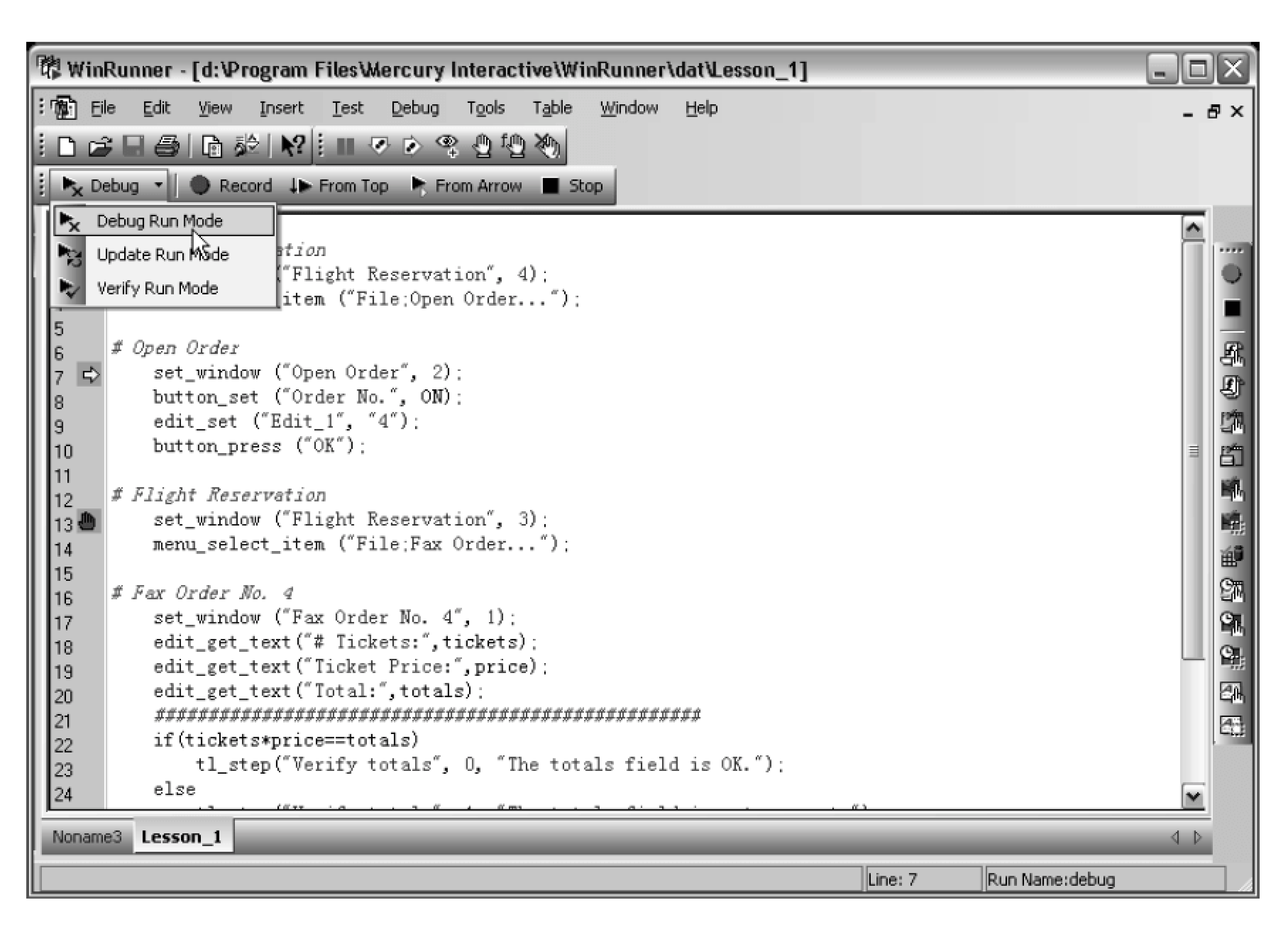


图 9-12 设置调试运行模式

4. 运行测试

在运行测试脚本前，将运行模式设置为验证运行模式(Verify Run mode)，即可运行脚本来测试应用程序。当 WinRunner 在运行中遇到检查点时，会将被测应用程序中的当前数据与以前捕捉的期望数据进行比较，不论发现任何不匹配，WinRunner 都会将目前情况捕捉下来作为真实的结果。

5. 检查测试结果

每次测试脚本运行结束后，WinRunner 将会执行结果显示在报告中。该报告描述所有在运行中所遇到的重要事件，例如检查点、错误信息、系统信息或是用户信息。如果发现在运行中有任何不匹配的检查点，可以在测试结果窗口中查看期望的和实际的结果，如图 9-13 所示。

6. 提交缺陷

如果一个测试脚本是由于所测试应用程序中的缺陷而导致执行失败的，可以直接从测试结果窗口中提取缺陷的相关信息。

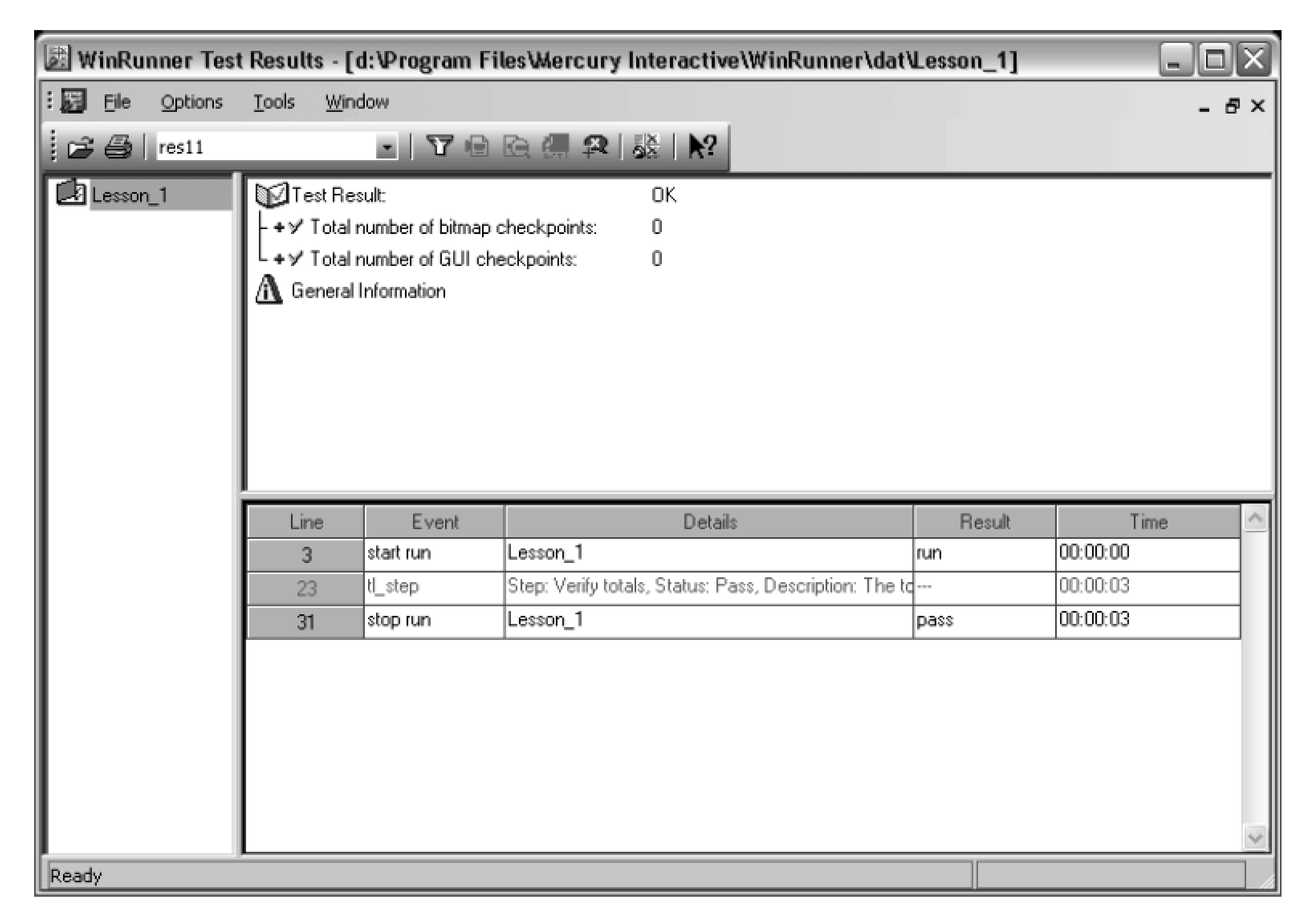


图 9-13　测试结果窗口

9.6.3　GUI Map

在讲解使用 WinRunner 创建测试项目之前，首先介绍 WinRunner 中的 GUI Map（图形用户接口映射）。通常，Windows 应用程序是由窗口、按钮、菜单、Text、Edit 等组件构成，在 WinRunner 中这些组件称为 GUI 对象（GUI object）。WinRunner 能在录制脚本或使用专用程序（例如 GUI SPY）的过程中，通过这些 GUI 对象的属性如 Class，Label，Width，Height，Handle 和 Enabled 等来识别它们。对于学过面向对象编程工具如 VB，VC 或 Delphi 等的读者来说，这些对象的属性是非常熟悉的。

WinRunner 识别 GUI 对象的过程非常简单。例如，当 WinRunner 识别一个 OK 按钮时，会记录这个按钮的属性信息，如所属的窗口（如属于 Open 窗口中的 GUI 对象），所属的 Class（如 push-button），按钮的文字卷标（如 OK 按钮）等。但对于 Width，Height，Handle 或其他属性一般会忽略掉。WinRunner 在识别 GUI 对象后，会将这些属性描述信息保存在 GUI 文件中，这种文件就称为 GUI Map 文件。这样，当运行测试脚本时，WinRunner 会根据测试脚本中的 GUI 对象的标识如 label，在相应的 GUI 文件中找到对象在 Windows 操作系统中注册的标识，然后通过标识找到相应的 GUI 对象，再对其进行操作。如果 GUI Map 文件不存在或内容出现错误，会造成测试脚本无法正确运行。

由于 GUI Map 在 WinRunner 应用的体系结构中处于核心地位，因此理解和掌握 GUI Map 的概念和正确使用 WinRunner 显得至关重要。

1. GUI Map File 模式

在创建测试工程时，应首先确定 GUI map file 模式，这可以在 General Options 对话框（图 9-10）中进行选择。在 WinRunner 中，有 GUI Map file per test 和 Global GUI Map file 两种 GUI map file 模式。下面分别简单介绍之。

（1）Global GUI Map file 模式

在 WinRunner 中，一般来说不同的测试脚本所测试的 GUI 对象是不完全相同的，但

如果为每组测试都准备一个 GUI Map 文件，在某些情况下则显得冗余。因此，共享 GUI Map 文件是一种很好的方法。Global GUI Map file 模式采用的便是这样一种思想。在这种模式下，可以多个测试脚本共享一个 GUI Map 文件。在开启测试脚本时，同时加载所使用的 Global GUI Map 文件即可。

产生 Global GUI Map 文件的方法是在录制测试脚本前，让 WinRunner 全面学习被测软件的所有 GUI 对象，一般采用 WinRunner 提供的 RapidTest Script Wizard 功能来完成。这种模式适合已经熟悉了 WinRunner 使用的用户。

在采用 Global GUI Map file 模式的情况下，如果在录制脚本时，出现了一些新的 GUI 对象，WinRunner 会自动将其识别出来，并保存在临时的 GUI 文件中。在 WinRunner 的 GUI Map Editor 窗口中，该临时文件显示为 * L0 ＜Temporary＞文件，如图 9-14 所示。在这种情况下，应注意将临时 GUI 文件合并到全局的 GUI Map 文件中，否则运行脚本时系统会提示找不到对象。

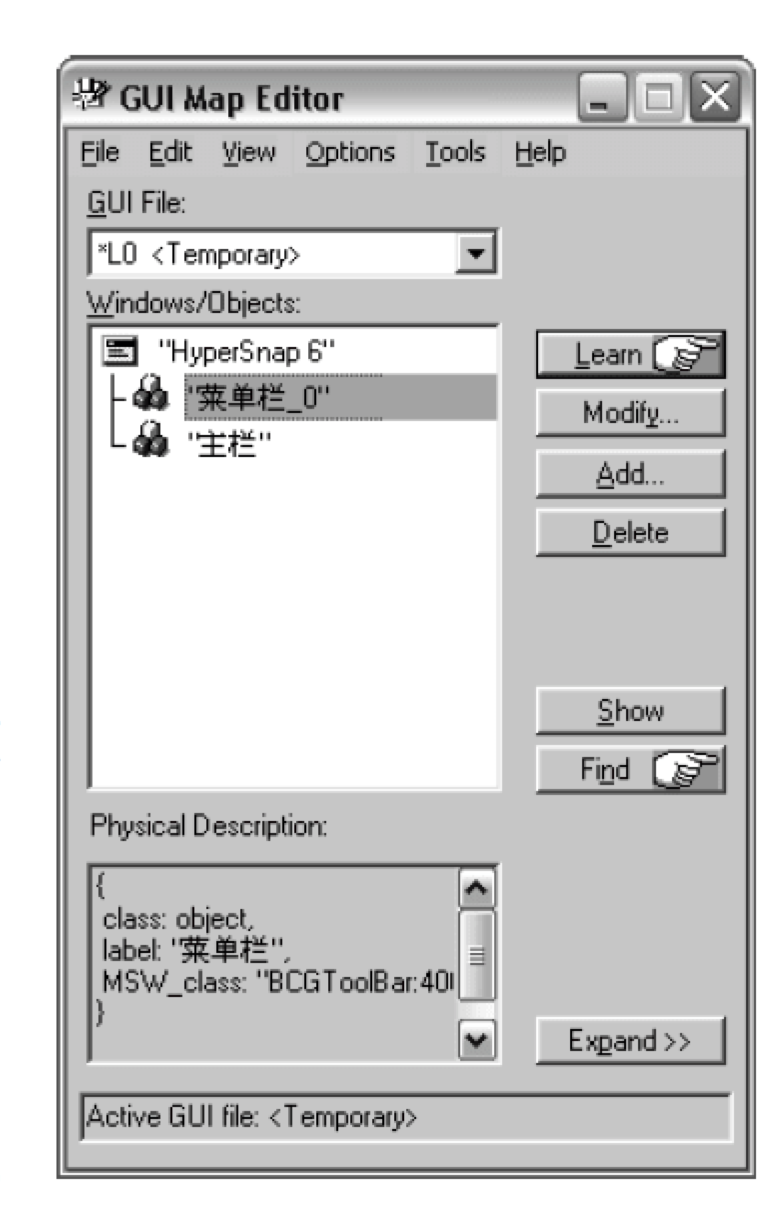


图 9-14　临时 GUI 文件

（2）GUI Map file per test 模式

在这种模式下，当用户新建立一个测试脚本时，WinRunner 会自动建立此测试脚本的 GUI Map 文件。当存储测试脚本时，WinRunner 会自动储存 GUI Map 文件。开启测试脚本，WinRunner 会自动加载 GUI Map 文件。总之一切与 GUI Map File 有关的动作，都由 WinRunner 自动处理。因此，这种模式很适合 WinRunner 的初学者。

2. 使用 RapidTest Script Wizard 学习 GUI 对象

下面以一个例子演示如何使用 RapidTest Script Wizard 自动学习被测软件的 GUI 对象以生成 GUI Map 文件。使用的被测软件是 WinRunner 自带的 Flight 软件，该软件为机票预定系统。

（1）在 Windows 的“程序”菜单中，单击 WinRunner→Sample Applications 文件夹中的 Flight 4A 选项，如图 9-15 所示。



图 9-15　启动 Flight 4A 命令

（2）打开的 Flight 4A 登录对话框如图 9-16 所示，在对话框中输入代理用户名(Agent Name)，名称任意，但必须不小于 4 个英文字符；密码(Password)mercury，然后单击 OK 按钮。

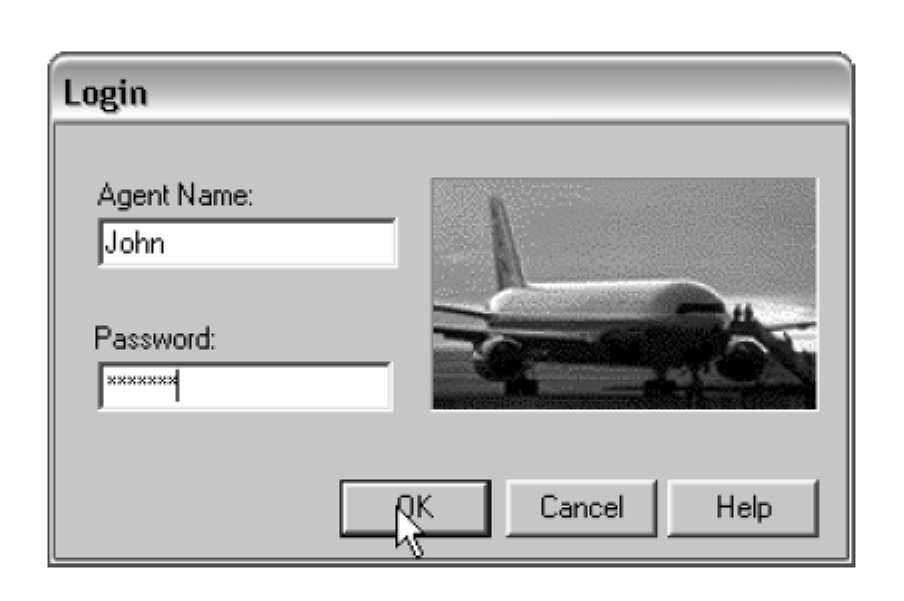


图 9-16 Flight 4A 登录对话框

(3) 验证正确后,将打开系统主界面 Flight Reservation 窗口,如图 9-17 所示。



图 9-17 Flight 4A 系统主界面

(4) 运行 WinRunner,创建一个新的测试。启动 RapidTest Script Wizard,打开 RapidTest Script Wizard 对话框,如图 9-18 所示,单击 Next 按钮,进入下一步对话框。

(5) 在打开的对话框(图 9-19)中,单击"手指"按钮,将鼠标移动到 Flight Reservation 窗口,如图 9-20 所示。注意,鼠标形状应是手指形状,且选中窗口的边框应处于闪烁状态。

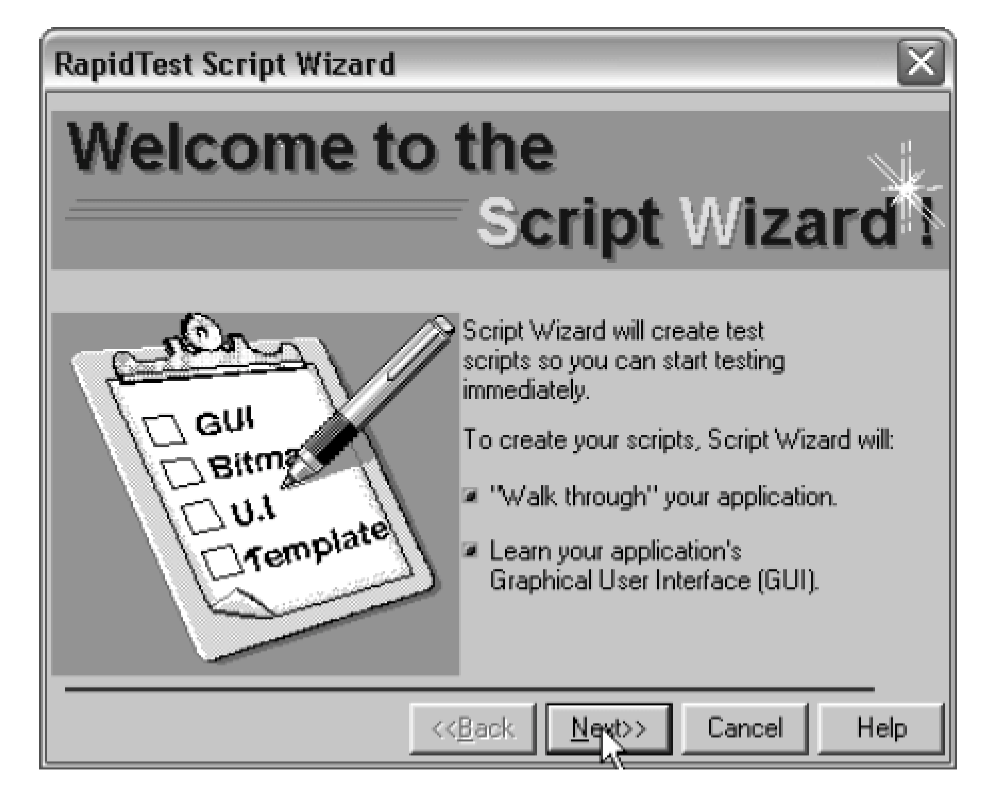


图 9-18 RapidTest Script Wizard 欢迎对话框



图 9-19 选择应用程序



图 9-20　Flight Reservation 窗口

（6）此时，图 9-19 对话框的 Window Name 文本框中将出现将被学习的窗口名称——Flight Reservation，如图 9-21 所示。

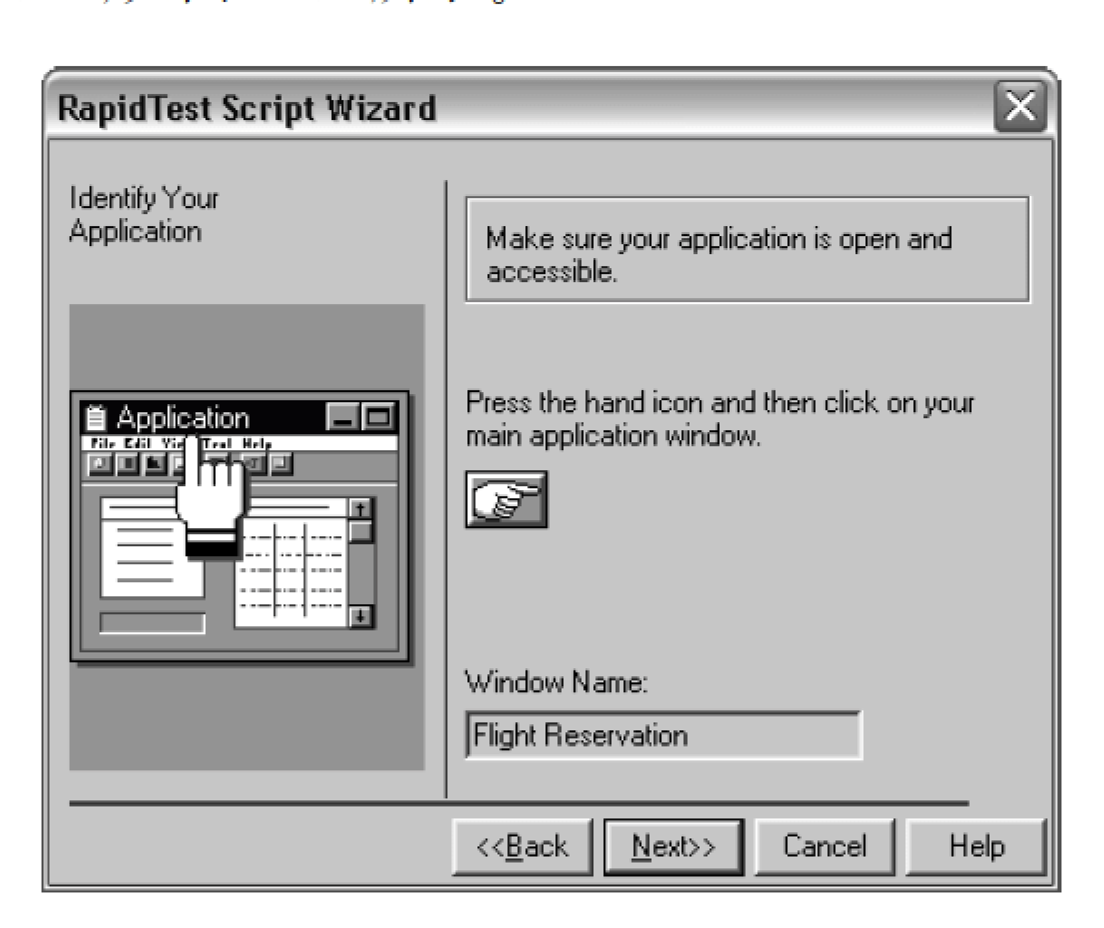


图 9-21　应用程序被识别

（7）单击 Next 按钮进行下一步操作，进入如图 9-22 所示对话框，取消所有复选框的选取。

（8）单击 Next 按钮进行下一步操作，进入如图 9-23 所示对话框。选中 Express 单选按钮，再单击 Learn 按钮。

（9）这时，Flight Reservation 窗口中的各个子窗口将开始闪烁，WinRunner 开始记录所有 GUI 的状态及相关信息，同时左上角将有个信息框显示工作状态。该过程可能会进行较长的时间，并且被测软件所包含的子窗口越多，识别所耗费的时间就越多。注意在识别的过程中，WinRunner 有时可能会报错。最常见的错误是 WinRunner 无法关闭 GUI 对象窗口，这种错误的原因可能是 WinRunner 向操作系统发出关闭窗口的消息，但

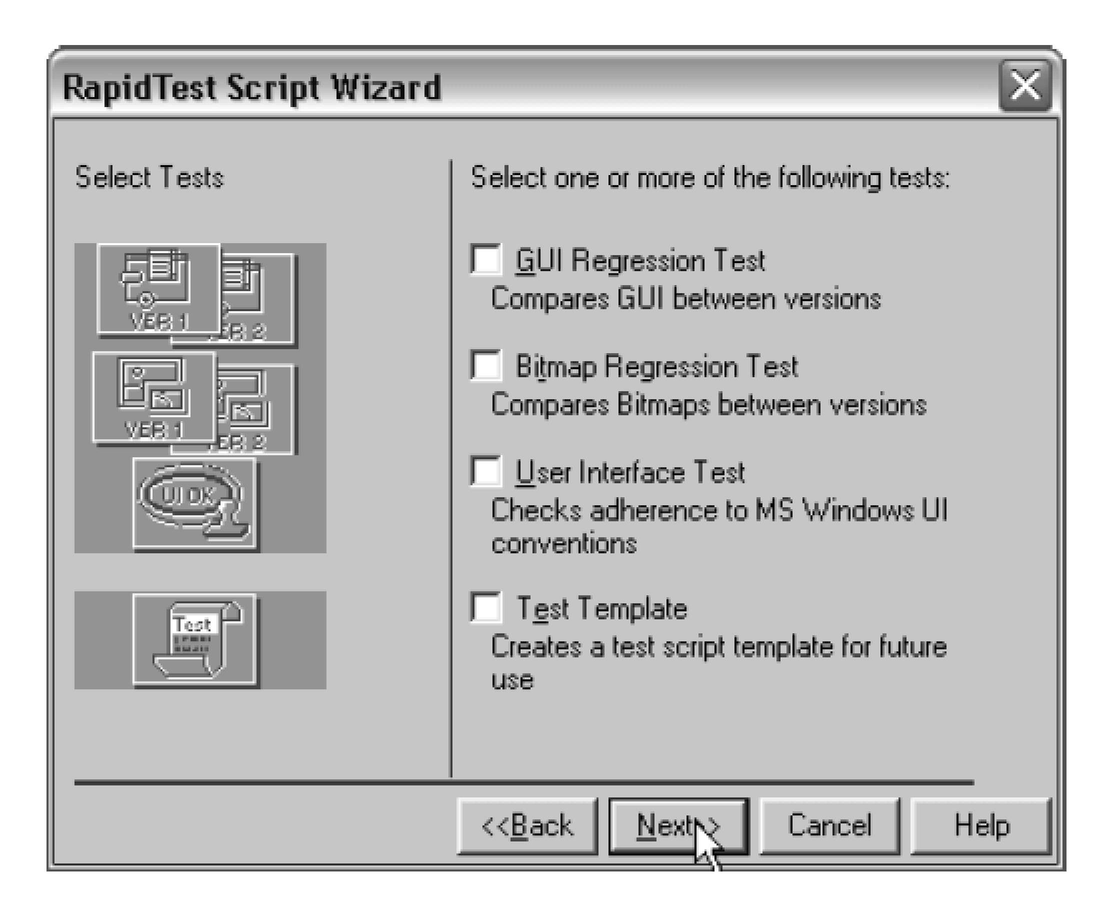


图 9-22　选择测试类型

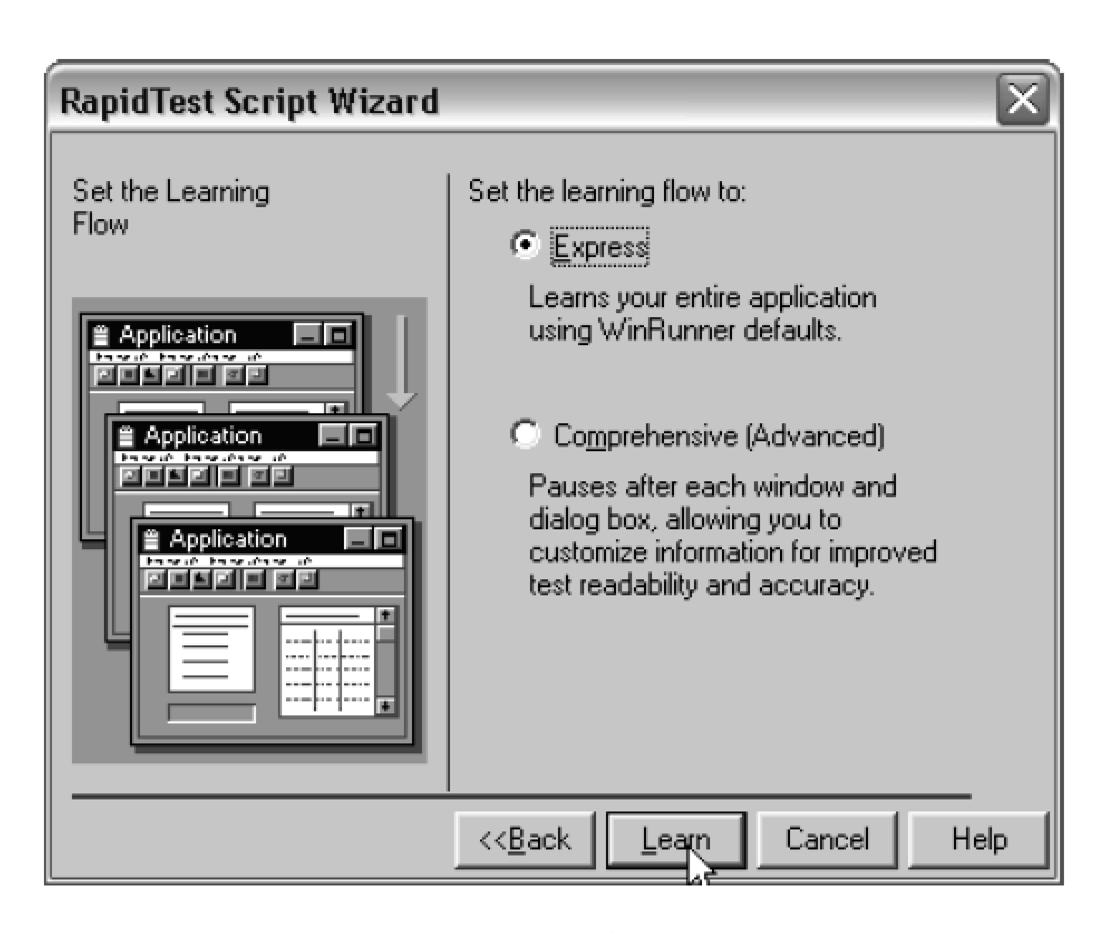


图 9-23　设置学习的流程

在规定的时间内没有响应。遇到这种情况，读者可以重新将识别的过程进行一次。为了减少此类错误的发生，建议在识别过程中尽量关闭其他无关的应用程序。

（10）当窗口停止闪烁时，出现如下提示对话框如图 9-24 所示，选中 NO 单选按钮，



图 9-24　提示对话框

单击 Next 按钮进行下一步。

(11) 出现保存 GUI Map file 路径对话框,如图 9-25 所示。在对话框中,可以修改其保存路径和 GUI Map 文件的名称。然后单击 Next 按钮进入下一步。

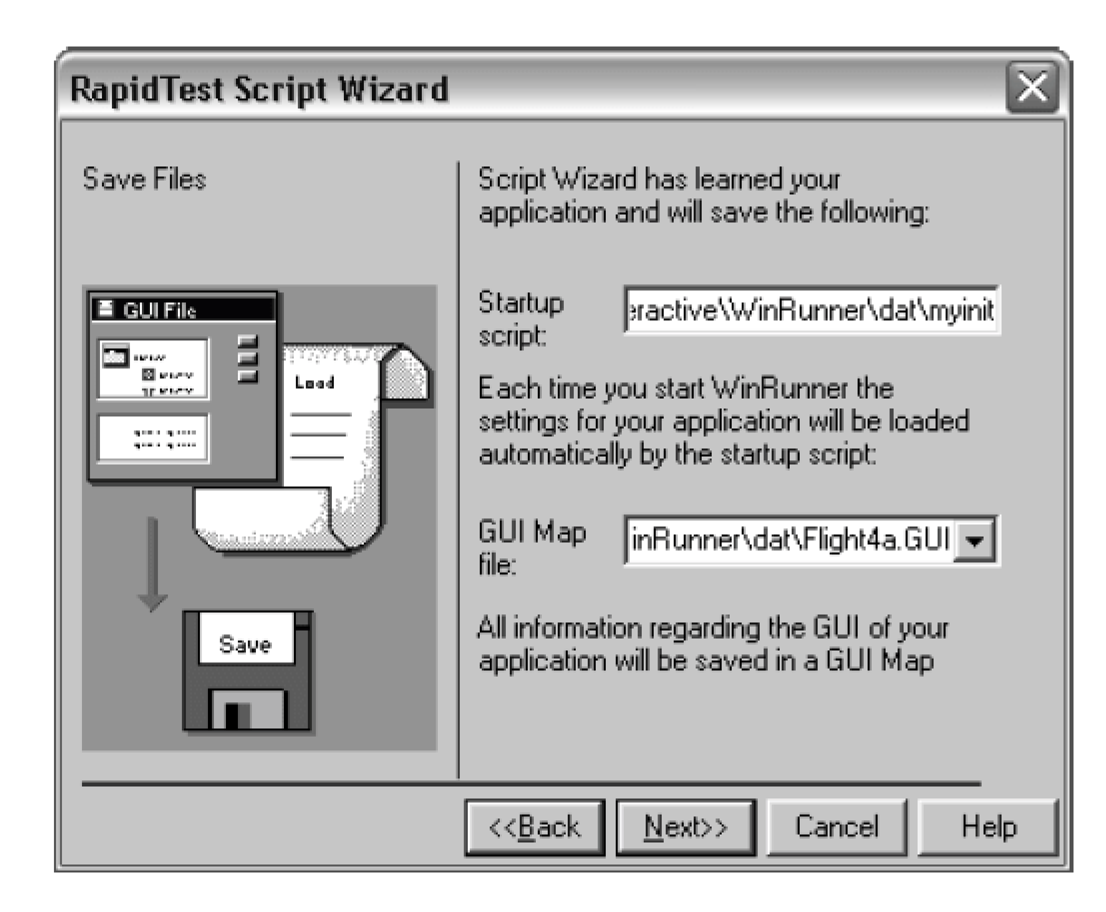


图 9-25 保存 GUI Map 对话框

(12) 出现 Congratulations 对话框,如图 9-26 所示,单击 OK 按钮。至此,识别过程完成。

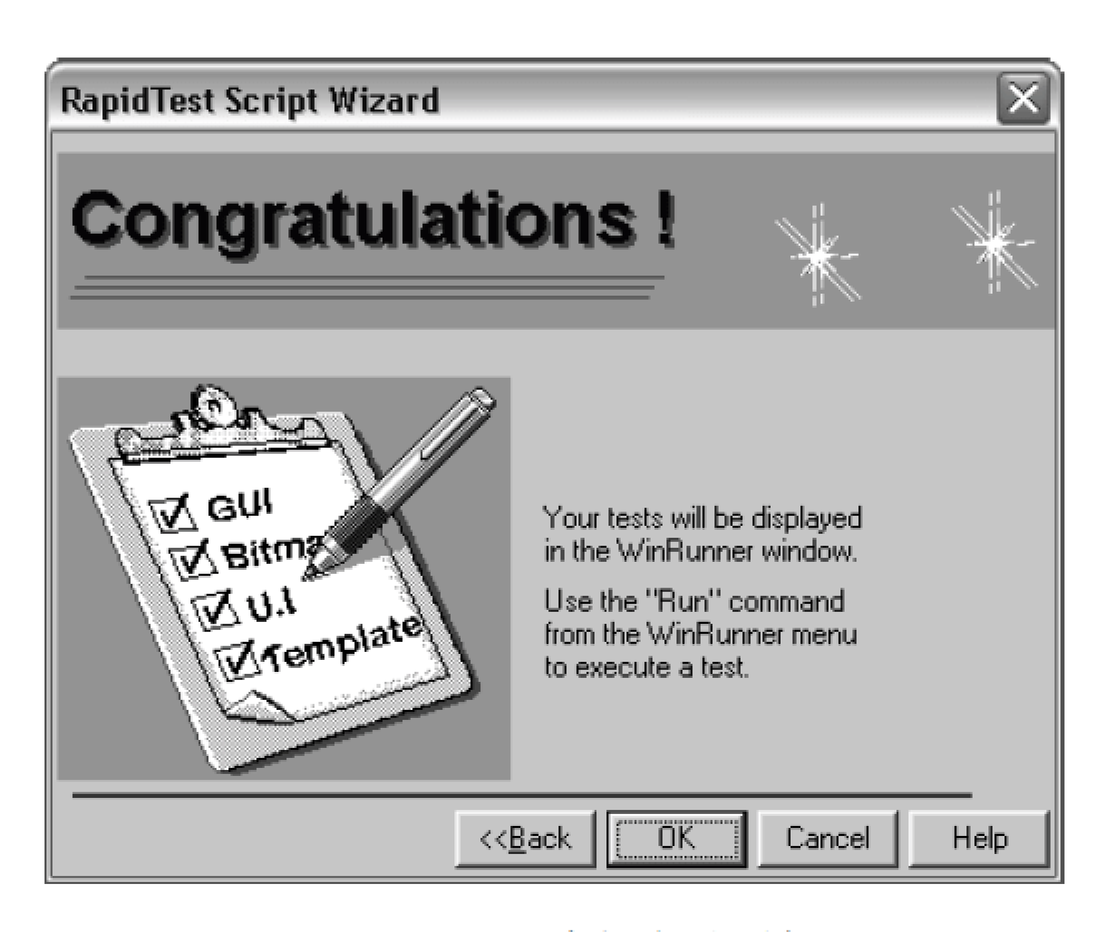


图 9-26 识别成功对话框

9.6.4 创建测试

在 WinRunner 中,生成测试工程的方法有自动录制脚本和手工编写脚本两种。后者使用方式灵活,功能较强,但只适合对 WinRunner 使用较熟练的测试人员。对初学者,一般使用自动录制脚本方式进行入门的学习。在实际应用过程中,多为将这两种方法的结合,即先使用自动录制脚本方式创建测试工程,然后再用手工添加所需脚本。本节主要学习如何使用上下文相关模式录制脚本(有关模拟模式,读者可以参看检查点测试中的图像检查点)。我们使用的被测软件是 WinRunner 自带的 Flight 系统。

1. 录制测试脚本

测试脚本录制步骤如下。

(1) 运行 WinRunner，单击 File 菜单中的 New 选项，创建一个新的测试。单击 File 菜单中的 Save 选项，或者工具栏上的"保存"按钮将其保存，文件名为 sample_1。需要注意的是，保存脚本文件时，WinRunner 将自动创建一个名为 sample_1 的文件夹，并将脚本文件保存在其中，如图 9-27 所示。

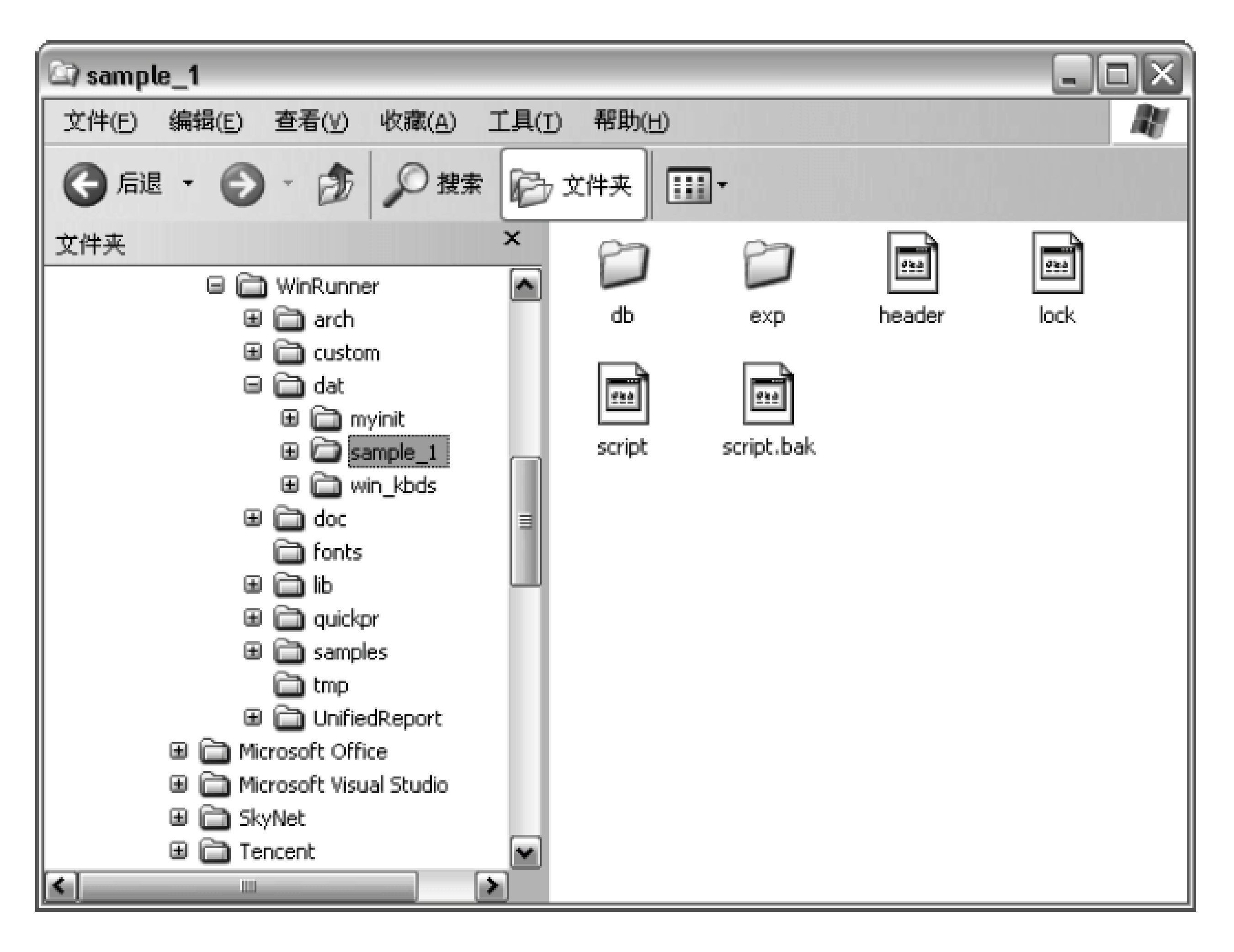


图 9-27　sample_1 文件夹

(2) 运行 Flight 4A，参看图 9-15～图 9-17。

(3) 单击 Test 菜单中的 Record-Context Sensitive 选项，开始以"上下文相关"(Context Sensitive)模式录制一个脚本。

(4) 返回 Flight Reservation 窗口，单击 File 菜单中的 Open Order 选项，如图 9-28 所示。

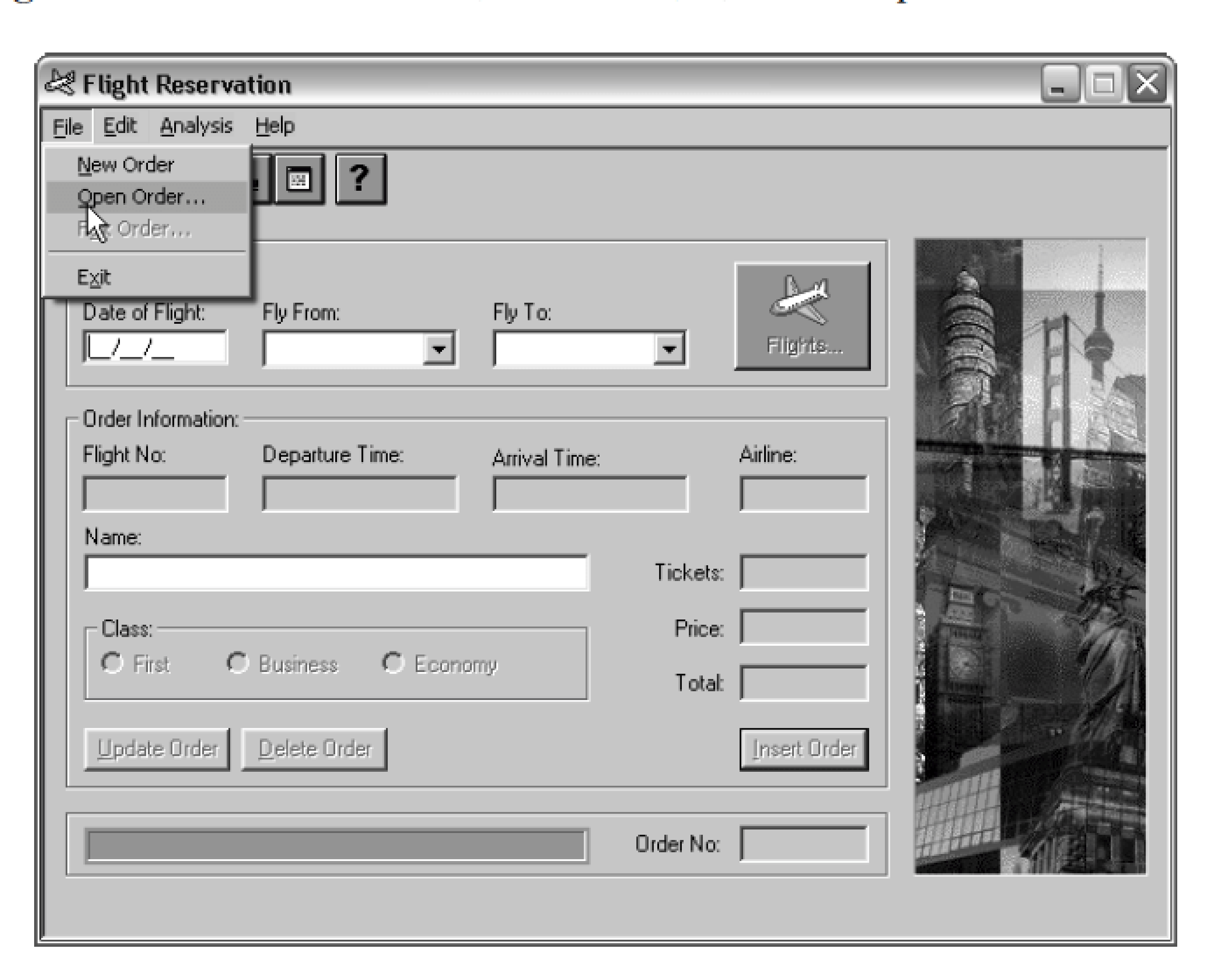


图 9-28　打开 Open Order 对话框命令

(5) 在打开的 Open Order 对话框，如图 9-29 所示，选中 Order No. 复选框，在文本框中输入数字 1(表示打开 1 号订单)，单击 OK 按钮。

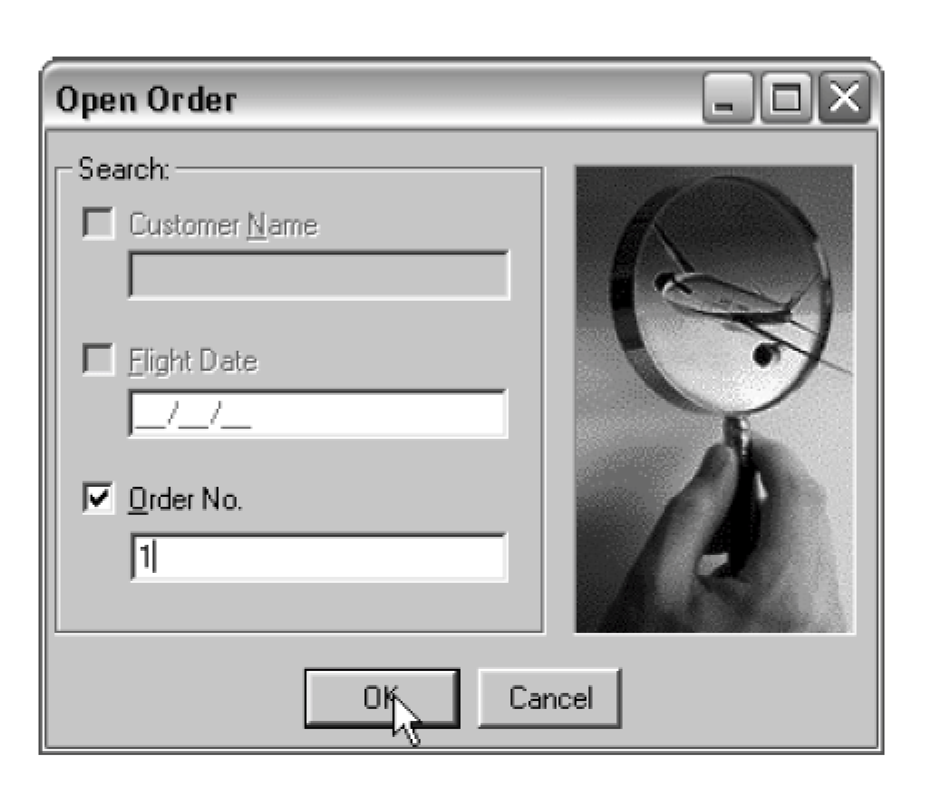


图 9-29　Open Order 对话框

(6) Flight Reservation 窗口中将出现所选中记录的内容，如图 9-30 所示。



图 9-30　订单记录窗口

(7) 单击工具栏上的“停止”按钮，停止录制。

(8) 保存录制的脚本。

2. 测试脚本简介

WinRunner 的测试文件是由测试脚本语言 TSL(Test Script Language)组成的文本文件。下面以 sample_1 脚本文件为例进行介绍，如图 9-31 所示。

```
# Flight Reservation
    win_activate ("Flight Reservation");
    set_window ("Flight Reservation", 4);
    menu_select_item ("File;Open Order...");

# Open Order
    set_window ("Open Order", 1);
    button_set ("Order No.", ON);
    edit_set ("Edit_1", "1");
    button_press ("OK");
```

图 9-31　测试脚本示例

(1) # Flight Reservation 语句是一条注释语句，在 WinRunner 中，注释语句均以 # 号开始，注释语句不被系统执行，只起说明作用。WinRunner 在打开新的窗口时，都会自动增加一行注释。

（2）在 win_activate ("Flight Reservation")中，win_activate()函数激活一个窗口，其参数为窗口名称，如本例中的"Flight Reservation"。

（3）每次打开新的窗口时，WinRunner 都会添加一个 set_window()函数，例如 set_window ("Flight Reservation", 4)，Flight Reservation 为窗口的名称，4 为延时的时间，单位为秒。

（4）menu_select_item ("File; Open Order...")即打开 File 菜单中的 Open Order 选项。

（5）单击一个对象时，WinRunner 会分配给它一个逻辑名称，一般为这个对象的文本标签(text label)。例如，选中窗口中的复选框按钮，WinRunner 将生成语句：button_set ("Order No.", ON)，其中 Order No 为这个对象的逻辑名称，ON 为复选框被选中。

（6）当从键盘输入时，WinRunner 会将产生一个 Type(类型)、obj_type(对象类型)或一个 edit_set()描述于脚本中。例如本例，在 Order No. 文本框中输入数字，则 WinRunner 会生成语句：edit_set ("Edit-1", "1")，其中 Edit-1 为文本框对象的逻辑名，1 为文本框的值为 1。

（7）button_press ("OK")表示单击 OK 按钮，关闭 Open Order 对话框。

更多的帮助可以单击 WinRunner 中的 Help 菜单中的 TSL Onling Reference 选项进行参考。另外，用户可以在 WinRunner 的工作窗口中获得帮助。方法是在脚本中选中某个函数或关键字，然后按 F1 键。

3. 运行测试脚本

录制完脚本之后，可以运行测试脚本，由 WinRunner 模拟用户的操作。本节以上节录制的 sample_1 脚本为例，介绍如何执行测试脚本。

（1）启动 WinRunner 和 Flight 4A，方法如前所述。

（2）选择 File 菜单中的 Open 选项，或在工具栏上单击“打开”按钮打开 sample_1 测试脚本。

（3）在工具栏上选择 Verify 模式按钮 Verify。

（4）选择 Test 菜单中的 Run from Top 选项，或在工具栏上单击 From Top 按钮 From Top，系统将弹出一个运行测试对话框，如图 9-32 所示。

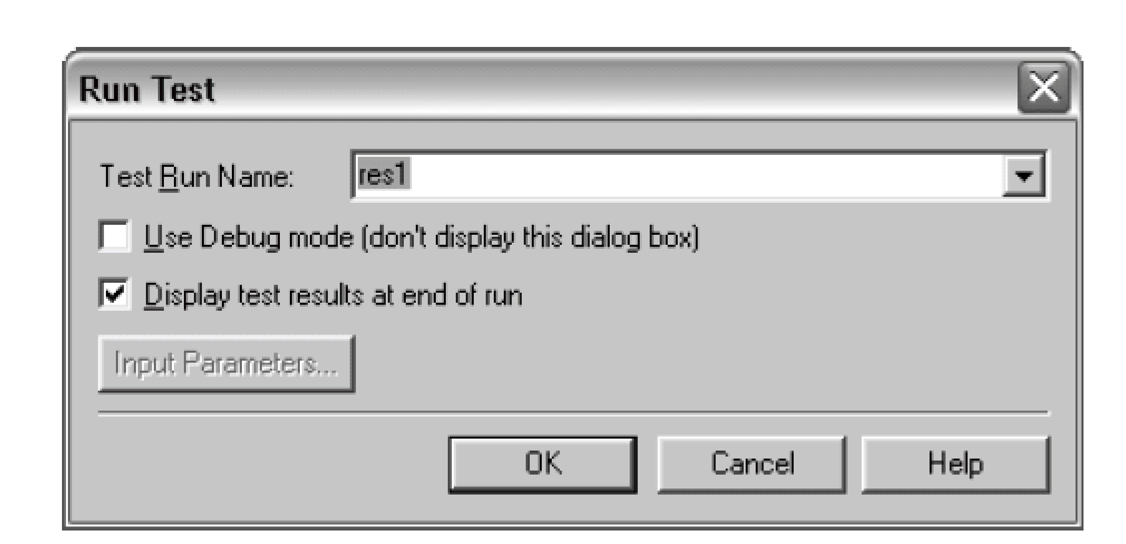


图 9-32　运行测试对话框

（5）在 Test Run Name 下拉列表中选择，系统默认为 res1。这个名称将作为文件夹保存在测试脚本所在文件夹下(sample_1 所在文件夹)。选中 Display test results at end

of run 复选框,表示运行完脚本后自动弹出结果窗口。

(6) 单击 OK 按钮,运行测试。

(7) 运行结束,自动显示分析结果。

4. 分析脚本运行结果

sample_1 脚本运行结束后,系统将自动弹出一个结果显示窗口,如图 9-33 所示。单击工具栏上的图标按钮也可以弹出该窗口。下面简单说明该窗口的组成。

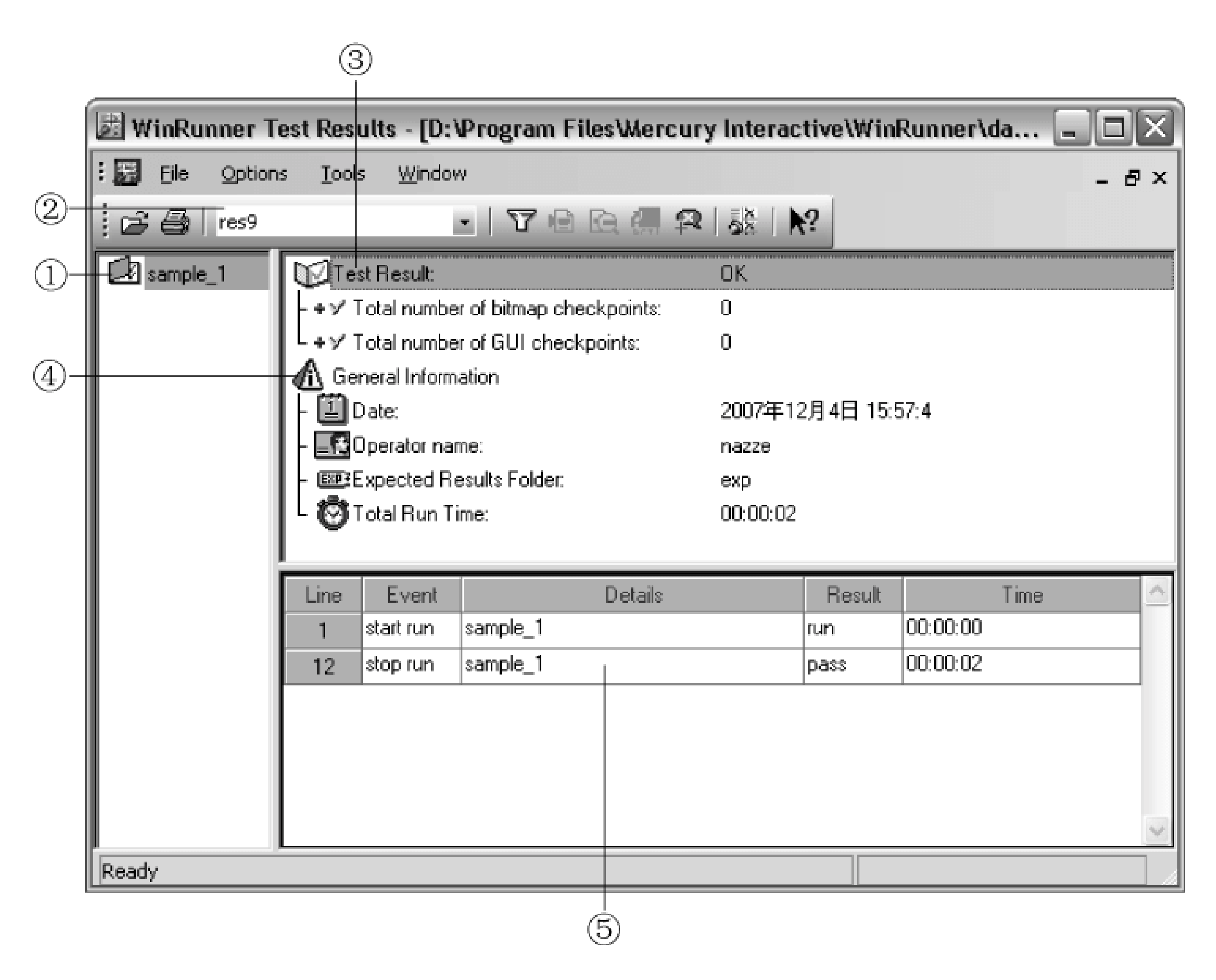


图 9-33　测试结果窗口的组成

该窗口各个部分的含义如下:

① 显示当前测试脚本的名称。

② 显示当前测试结果路径名称。WinRunner 中测试结果一般是放在一个形如 res# 的文件夹中,其中 # 代表测试的次数,例如图中示例表明是第 9 次测试。在下拉列表框中选择 res8,则显示如图 9-34 的窗口,表明第 8 次测试出现了错误,同时列出错误信息。

③ 显示测试运行的状态(通过或失败)。双击 Test Result 节点,可以折叠或展开该节点。

④ 显示本次测试的相关信息,如日期、操作者名称、异常结果所保存的文件夹和总运行时间。双击 General Information 节点,可以折叠或展开该节点。

⑤ 显示测试运行期间的主要发生事件、详细信息、运行结果、开始时间,以及每个发生事件在测试脚本中的行号。

9.6.5　GUI 检查点

通过前面的学习,我们对 WinRunner 的框架结构和测试流程已经有了初步的了解,也学习了如何录制脚本和运行测试。但到目前为止,并未使用测试脚本进行更多的实质性功能测试。本小节和之后的 9.6.6 和 9.6.7 小节将介绍 WinRunner 中的检查点

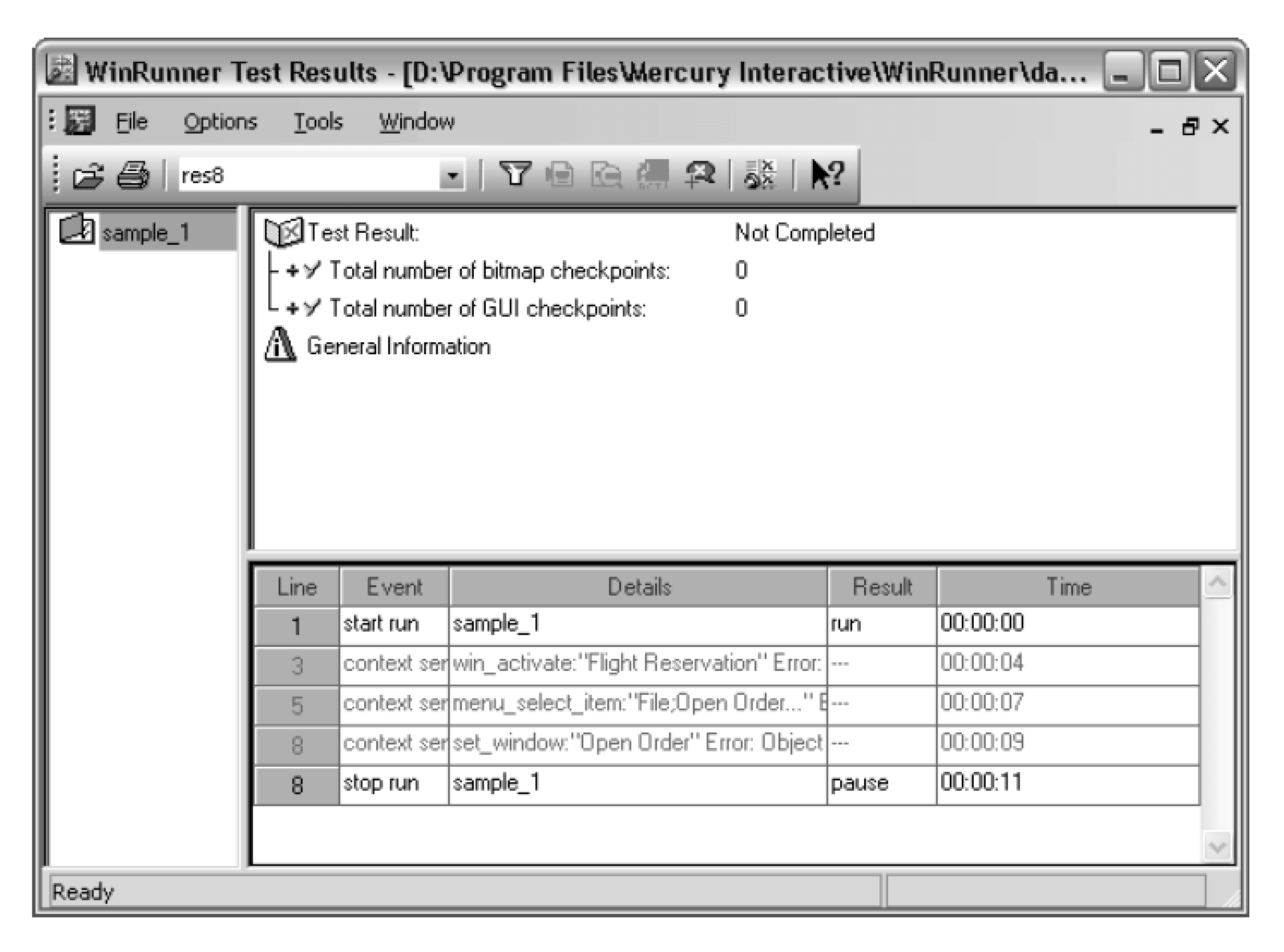


图 9-34　测试结果窗口示例

(Checkpoint)测试，以了解如何使用 WinRunner 进行较复杂的功能测试。

所谓检查点，是指在 WinRunner 的脚本中的一定位置，根据需要设置检查条件，以此判断程序的设计、运行是否正确。其主要的思想是判断程序中当前运行值与期望值是否匹配。WinRunner 中的检查点分为 4 类：GUI 检查点、文本检查点、图像检查点和数据库检查点。本书主要介绍前 3 类检查点。

在本小节，我们先介绍 GUI 检查点。可以使用 GUI 检查点来检查被测软件中 GUI 对象的属性，测试程序的功能是否正常。例如，可以检查某个 button 是否可用；某个编辑框的文本格式是否符合要求；某个系统的不同版本的 GUI 界面是否一致等。当 GUI 对象的属性值与预期值不符合时，就表示可能存在问题。下面用一个实例演示如何使用 GUI 检查点。

1. 建立 GUI 对象检查点

(1) 启动 WinRunner 和 Flight 4A。

(2) 创建新的测试文件 sample_2。

(3) 打开 GUI Map file 文件。选择 WinRunner 菜单栏 Tools 中的 GUI Map Editor 选项，打开 GUI Map Editor 窗口；选择 Files 菜单中的 Open 选项，打开 flight4a.GUI 文件。该文件是我们在前面的例子中建立的 Flight 4A 系统的全局 GUI 文件，如图 9-35 所示。

(4) 单击 WinRunner 菜单栏 Test 中的 Record-Context Sensitive 选项，开始以“上下文相关”(Context Sensitive)模式录制脚本。

(5) 返回 Flight Reservation 窗口，单击其 File 菜单中的 Open Order 选项，弹出 Open Order 对话框，如

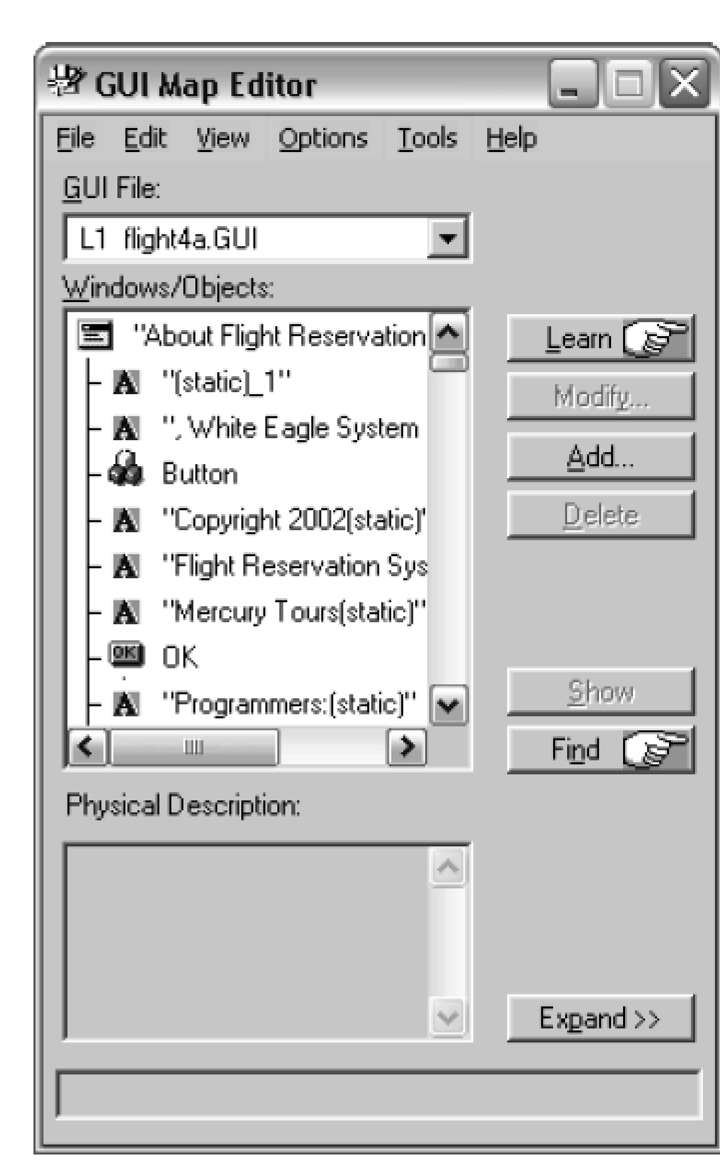


图 9-35　GUI Map Editor 窗口

图 9-36 所示。

(6) 在 Open Order 对话框中，选中 Order No.，输入数字 3，表示打开 3 号订单。

(7) 单击 WinRunner 菜单栏 Insert 中 GUI Checkpoint 级联菜单的 For Object/Window 选项，出现手形状光标，然后将此光标指向 Open Order 对话框中的 Order No. 文本框，如图 9-37 所示。

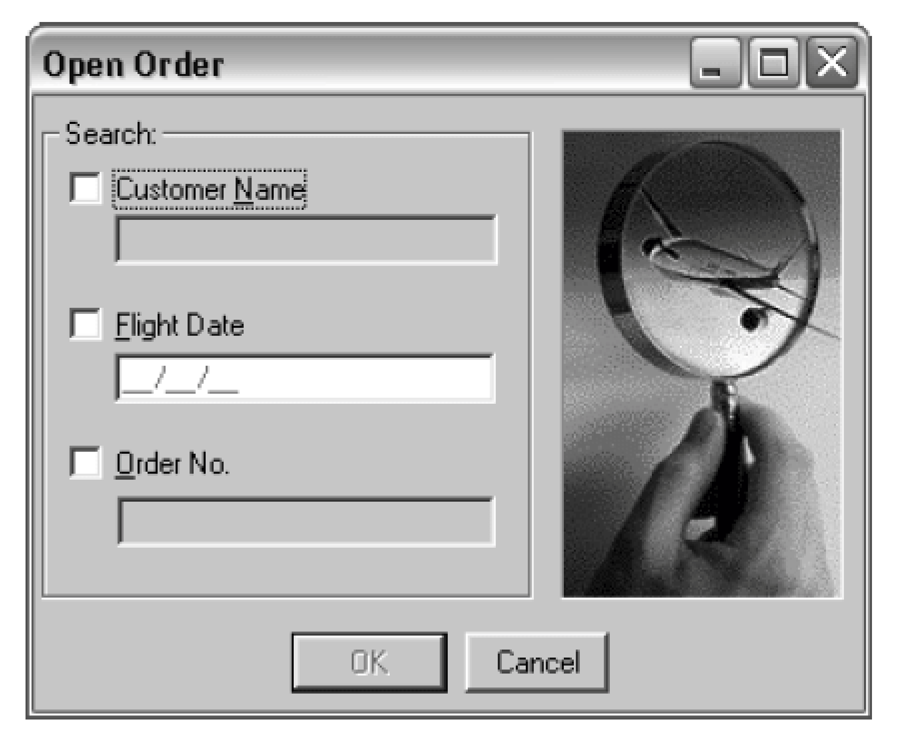


图 9-36 Open Order 对话框

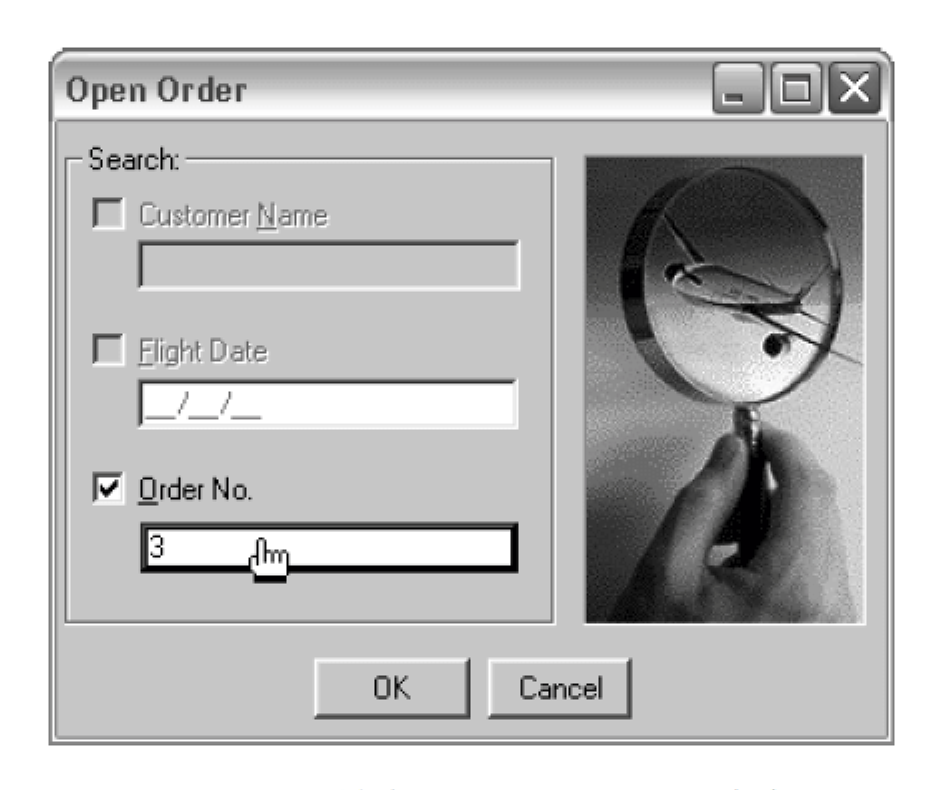


图 9-37 捕捉 Order No. 文本框

(8) 双击 Order No. 文本框，出现 Check GUI 对话框，如图 9-38 所示。该对话框中列出了 Open Order 对话框的检查清单。注意，如果单击 Order No. 文本框，则不会出现该对话框，而是直接在 WinRunner 中生成脚本语句。

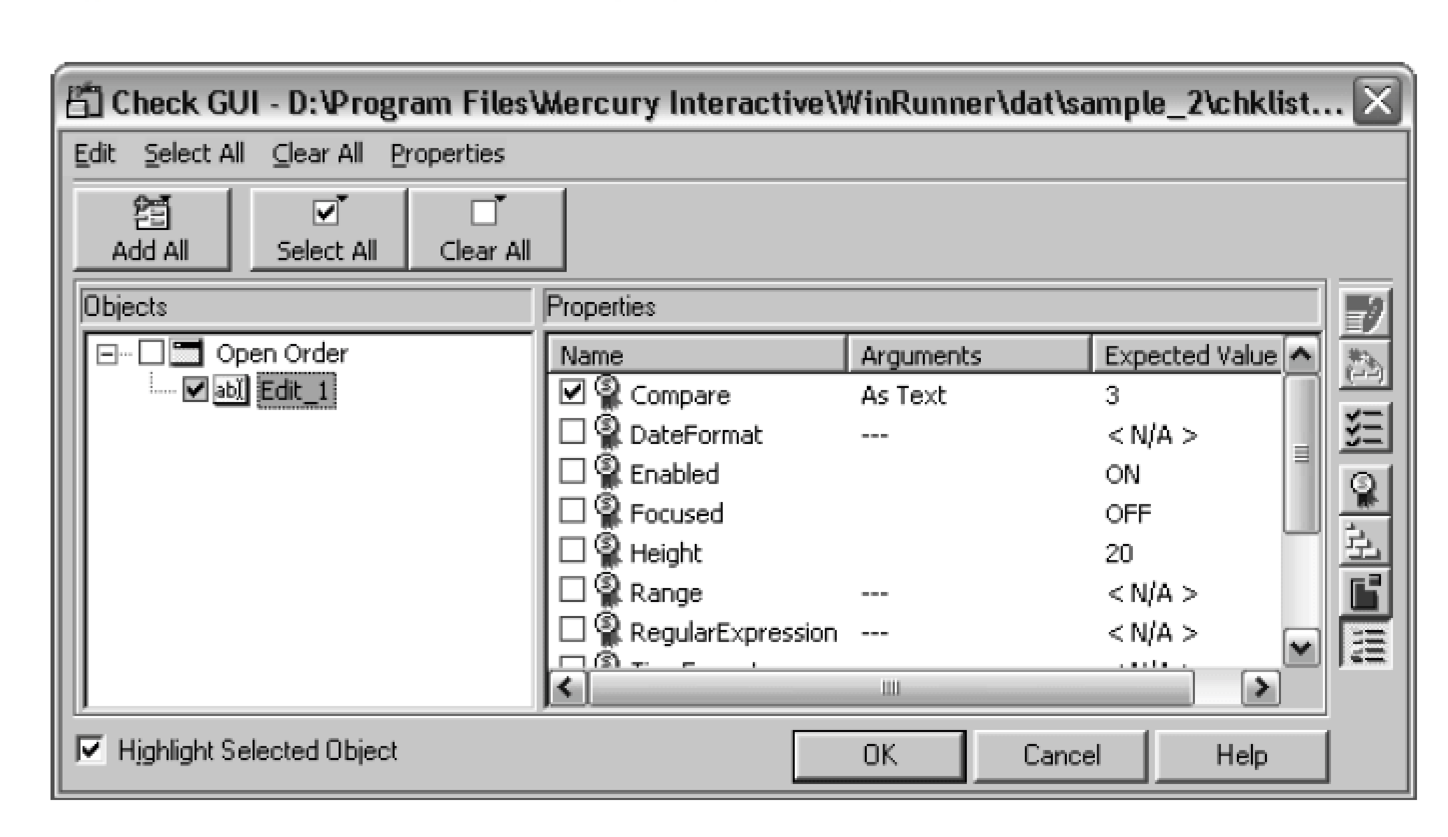


图 9-38 Check GUI 对话框

(9) Check GUI 对话框中的右下方是 Properties(属性)窗格。该窗格列出了当前 Edit_1 控件的属性和当前的期望值。选中该属性，即可编辑该属性的期望值。选中 Range 属性，单击其 Arguments 属性列，弹出 Check Arguments 对话框，如图 9-39 所示。在该对话框的 From 和 To 文本框中分别填入 1 和 5，表明该编辑框允许的范围值为 1～5。单击 Range 属性的 Expected Value 属性列，在文本框中填入 3。然后单击 OK 按钮，回到 Check GUI 对话框。

(10) 在 Check GUI 对话框中，单击 OK 按钮。此时 WinRunner 脚本窗格出现设置检查点语句 obj_check_gui，如图 9-40 所示。该函数中，list1.ckl 为检查清单文件，gui1 为

预期输出结果文件。

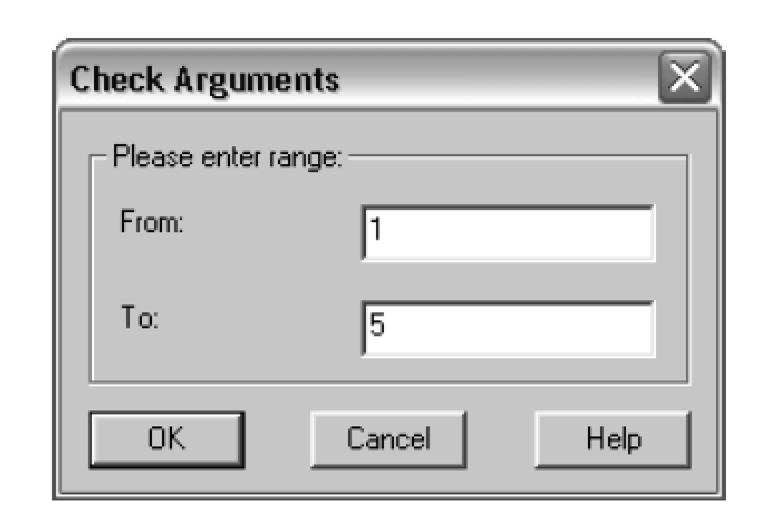


图 9-39　设置属性取值范围

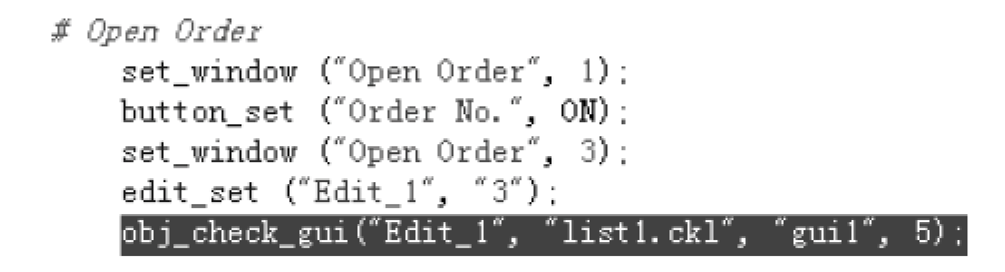


图 9-40　设置的检查点语句脚本

(11) 返回 Open Order 对话框，单击 OK 按钮，返回 Flight Reservation 窗口，如图 9-41 所示。



图 9-41　3 号订单记录窗口

(12) 单击 WinRunner 工具栏上的“停止”按钮 Stop，停止录制。

(13) 在工具栏上单击 From Top 按钮 From Top，运行该测试文件，测试顺利通过，并显示测试结果，如图 9-42 所示。

2. 设置错误 1

(1) 现在修改编辑框 Edit_1 的期望值。单击 WinRunner 菜单栏 Insert 中 GUI Checkpoint 级联菜单的 For Multiple Objects 选项，打开 Create GUI Checkpoint 对话框。单击 Open 按钮，打开 list1.ckl 检查清单文件，如图 9-43 所示。

(2) 在 Properties 窗格中，单击 Compare 属性的 Expected Value 属性列，在文本框中填入 4(在前面的录制脚本的过程中，其值为 3)，然后单击 OK 按钮，如图 9-44 所示。

此时 WinRunner 脚本窗口出现一条新的设置检查点语句 win_check_gui，如图 9-45 所示。该函数中，此时预期输出结果文件变成了修改过的 gui2。

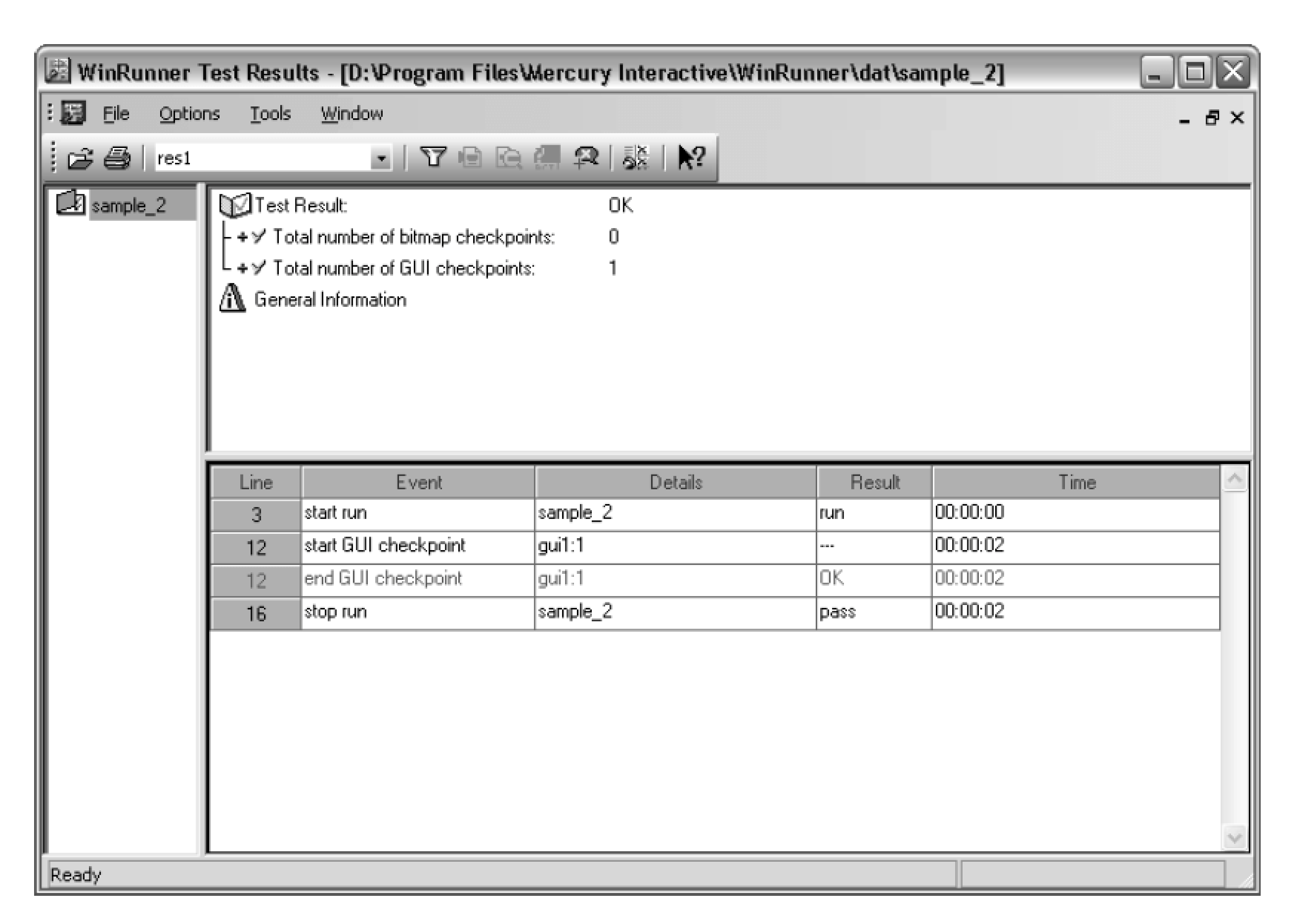


图 9-42　测试结果显示

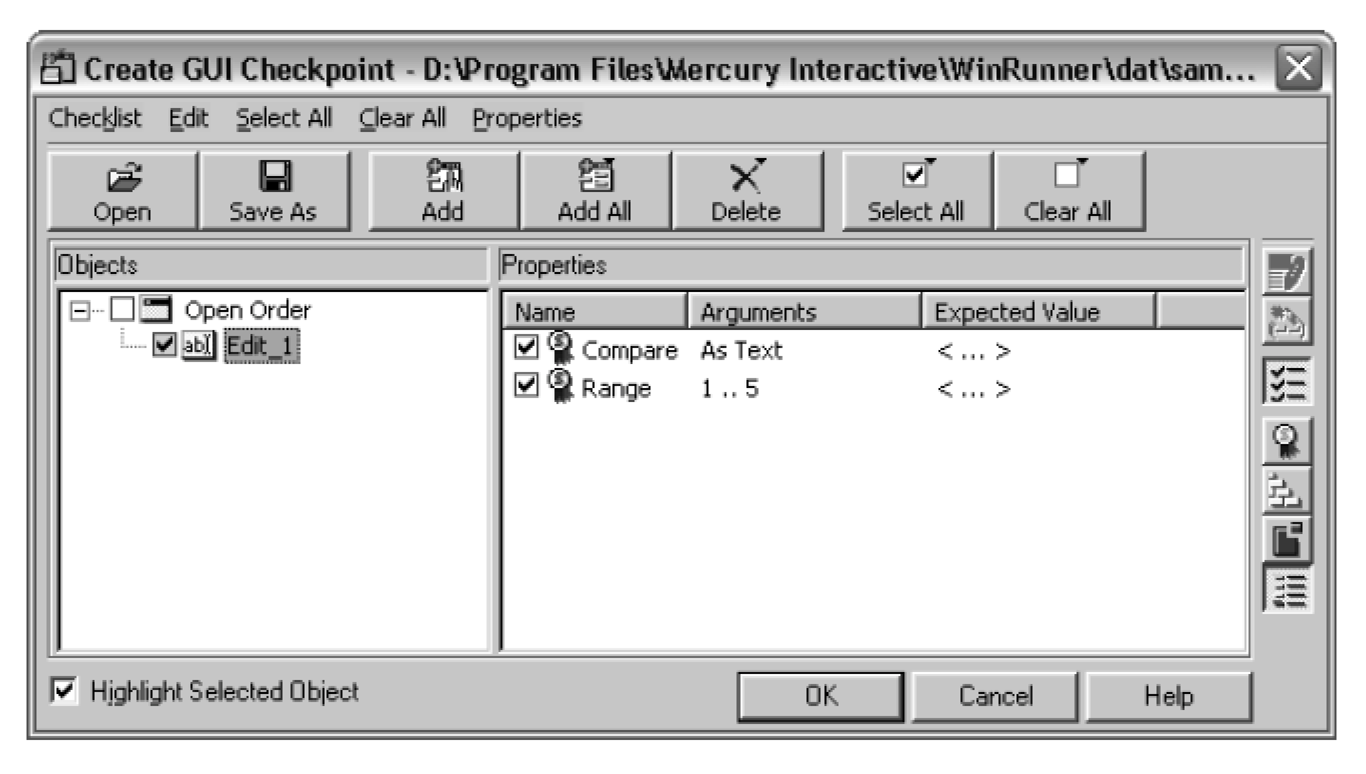


图 9-43　打开检查清单文件

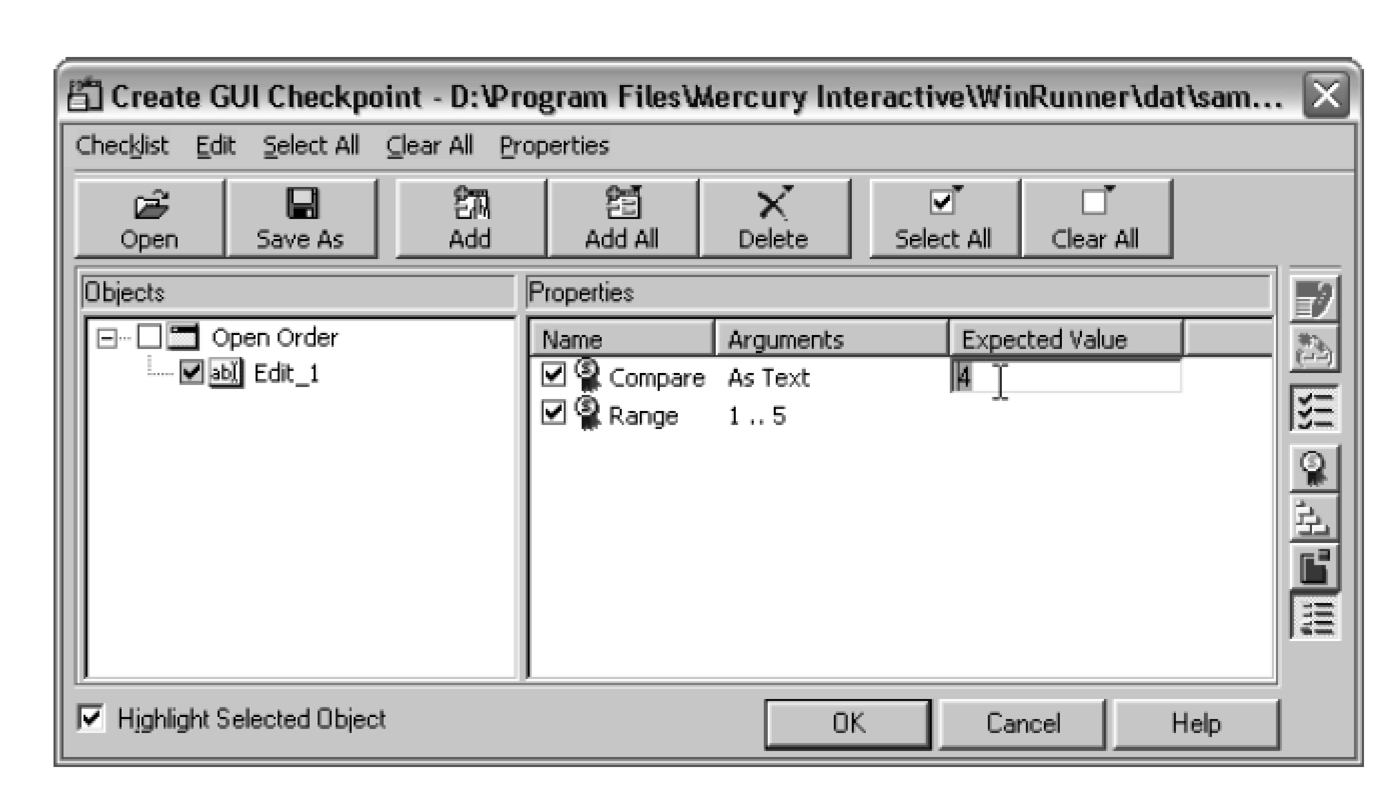


图 9-44　修改 Compare 属性的期望值

```
# Open Order
    set_window ("Open Order", 1);
    button_set ("Order No.", ON);
    set_window ("Open Order", 5);
    edit_set ("Edit_1", "3");
    obj_check_gui("Edit_1", "list1.ckl", "gui1", 1);
    win_check_gui("Open Order", "list1.ckl", "gui2", 1);
```

图 9-45　预期输出结果文件脚本

(3) 在 WinRunner 工具栏上单击 From Top 按钮 From Top 运行该测试文件,测试报错,并弹出报错对话框,如图 9-46 所示。

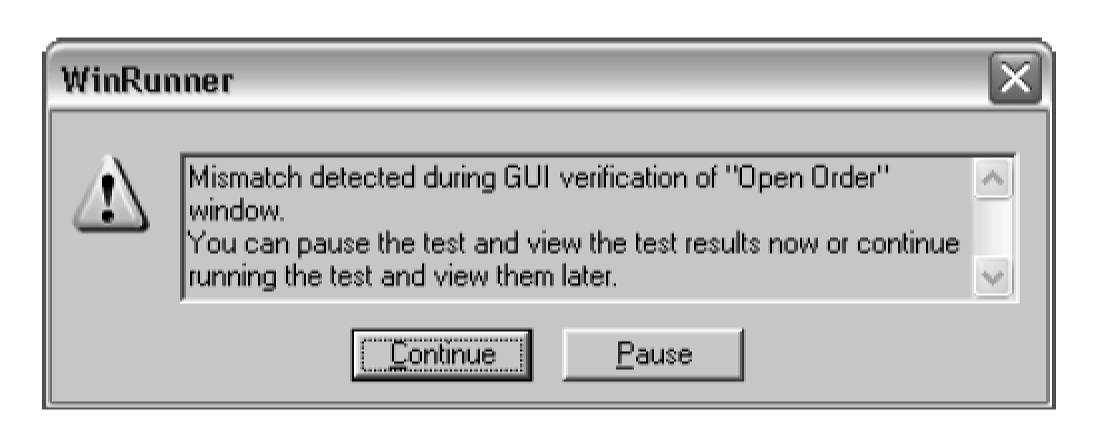


图 9-46　报错对话框

(4) 单击 Continue 按钮,完成测试。显示如图 9-47 所示测试结果。

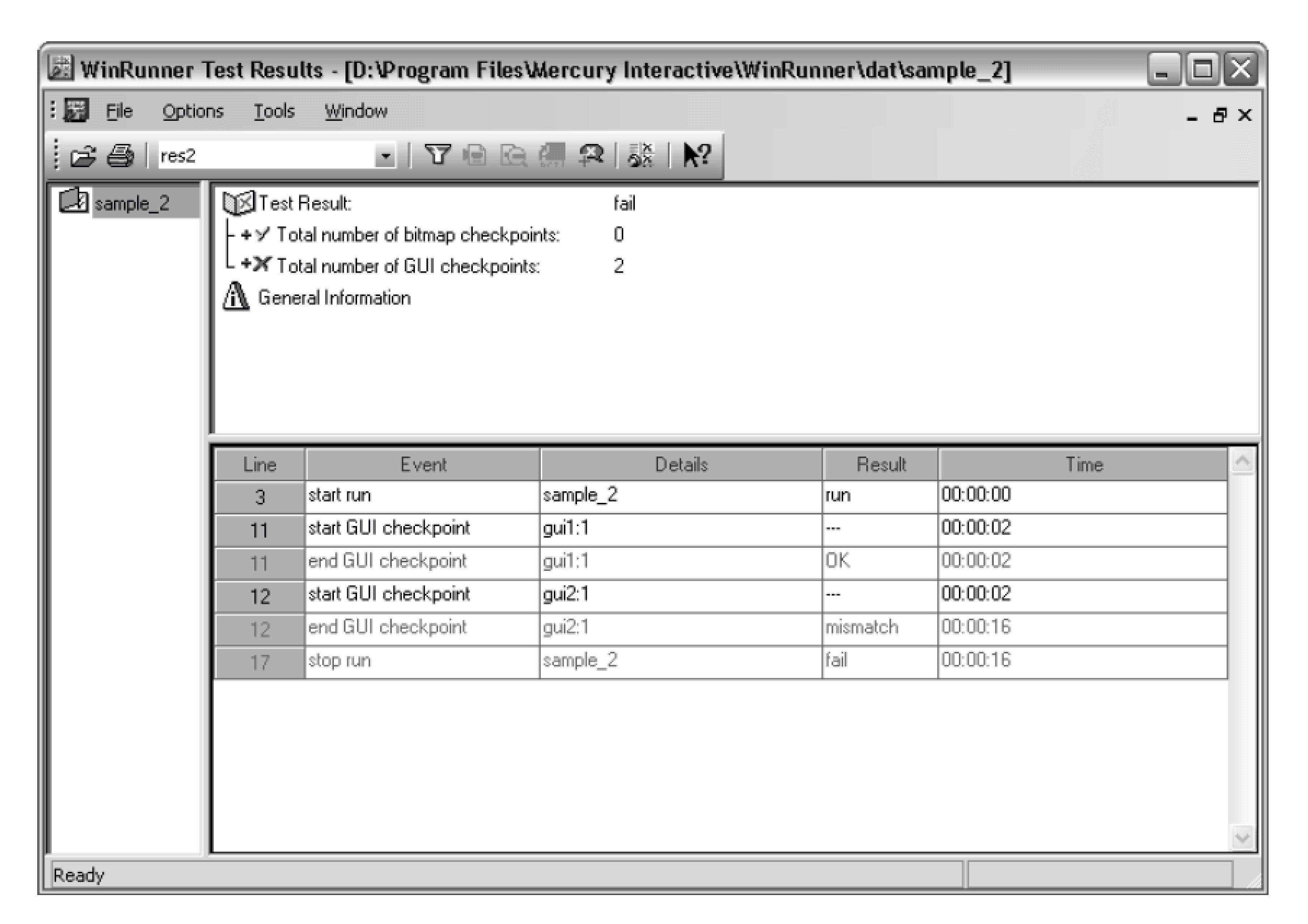


图 9-47　显示测试结果

在测试结果窗口的信息列表中双击红色显示的第 12 行错误信息,弹出 GUI Checkpoint Results(检查点错误信息)对话框,如图 9-48 所示。对话框内清楚地显示了产生错误的原因,是由于编辑框 Edit_1 的期望值与实际值不匹配。

3. 设置错误 2

(1) 重新修改编辑框 Edit_1 的期望值。单击 WinRunner 菜单栏 Insert 中 GUI Checkpoint 级联菜单的 For Multiple Objects 选项,打开 Create GUI Checkpoint 对话框。

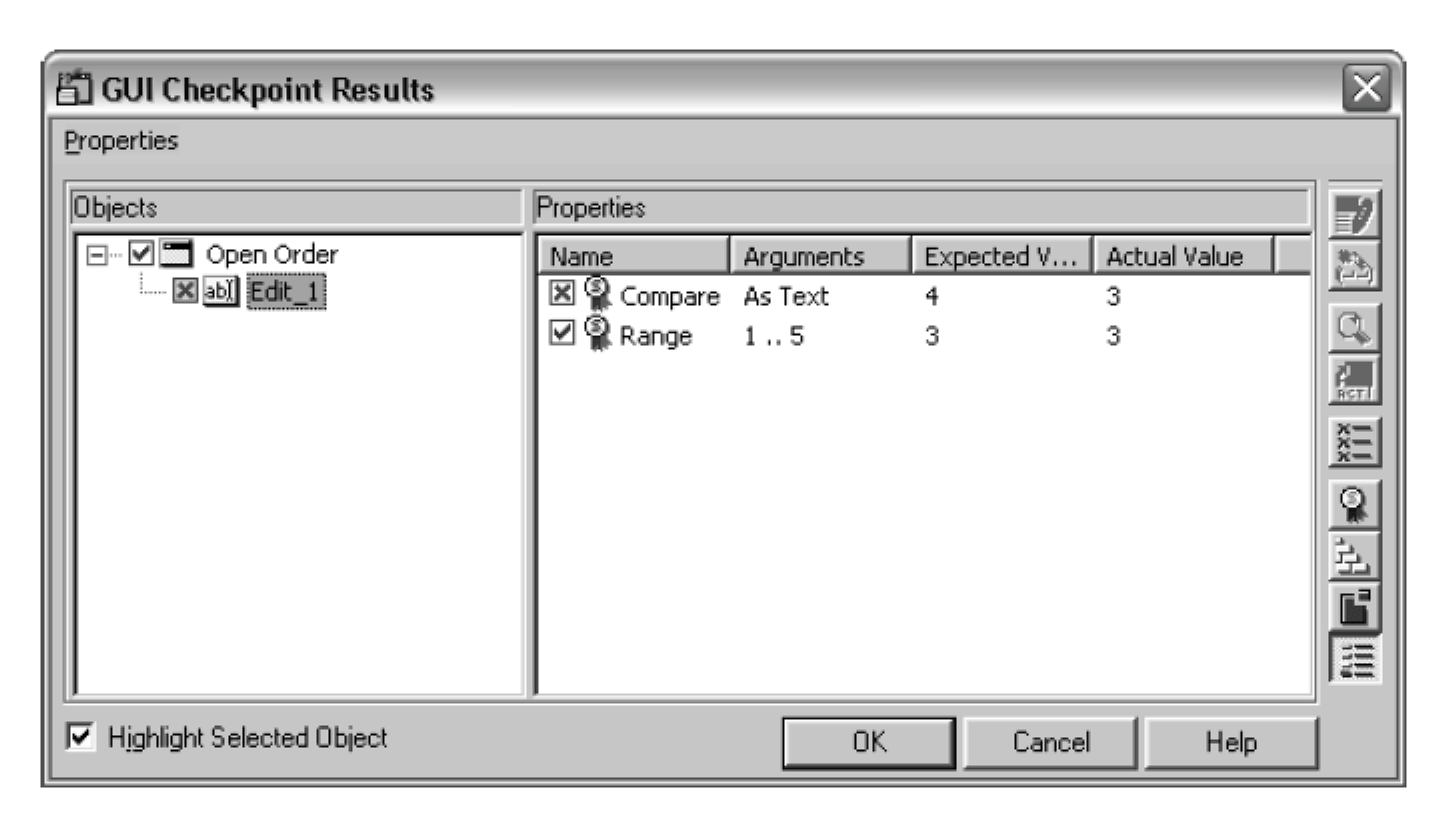


图 9-48 检查点错误信息对话框

单击 Open 按钮，打开 list1.ckl 检查清单文件，如图 9-43 所示。

(2) 在 Properties 窗格中，单击 Compare 属性的 Expected Value 属性列，在文本框中填入 6；单击 Range 属性的 Expected Value 属性列，在文本框中也填入 6，然后单击 OK 按钮，如图 9-49 所示。

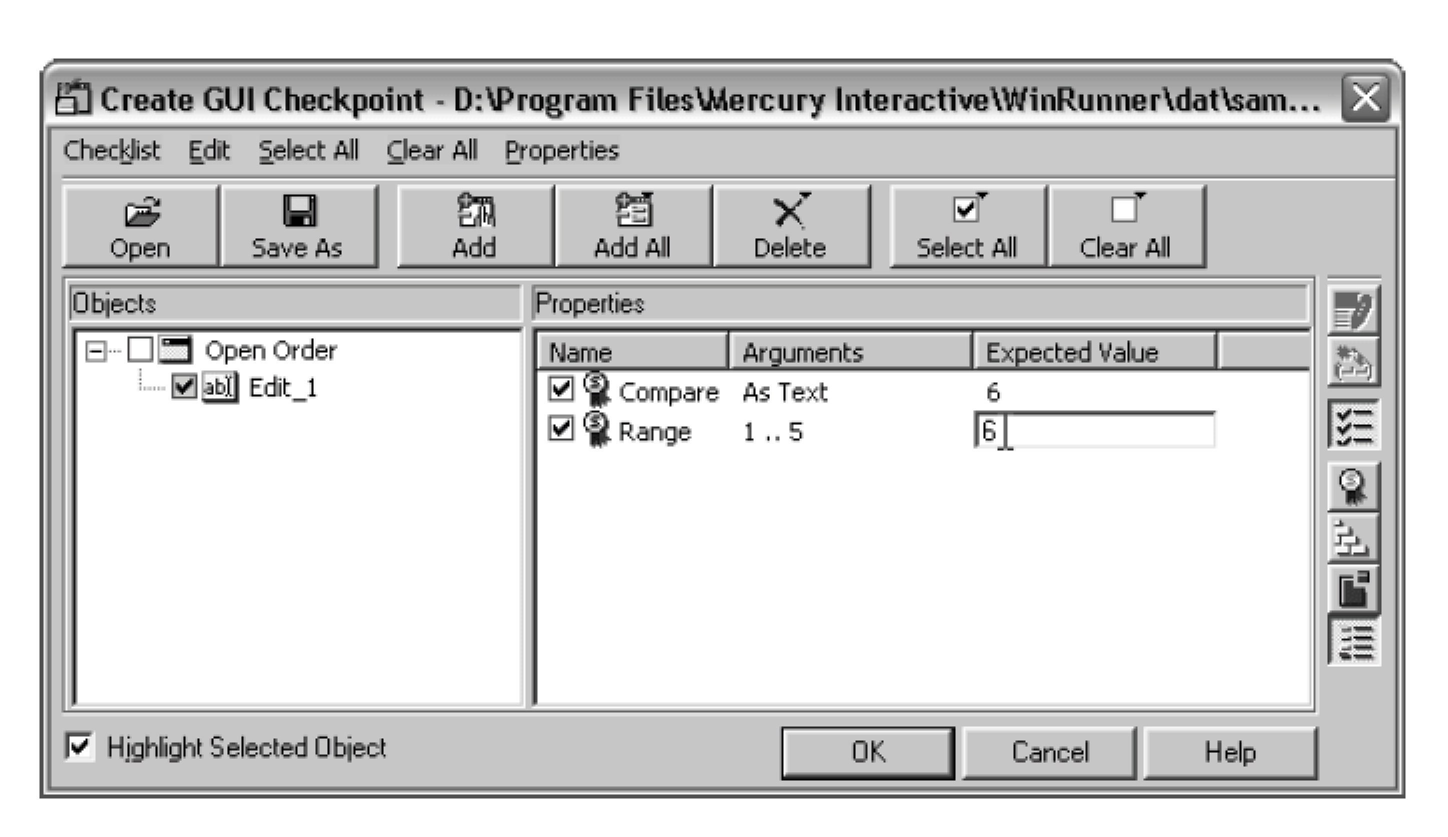


图 9-49 设置属性的期望值

此时，WinRunner 脚本窗口会出现一条新的设置检查点语句 win_check_gui，如图 9-50 所示。该函数中，此时预期输出结果文件变成了修改过的 gui3。

```
# Open Order
    set_window ("Open Order", 1);
    button_set ("Order No.", ON);
    set_window ("Open Order", 5);
    edit_set ("Edit_1", "3");
    win_check_gui("Open Order", "list1.ckl", "gui2", 1);
    win_check_gui("Open Order", "list1.ckl", "gui3", 1);
```

图 9-50 修改后的脚本(1)

删除语句 win_check_gui("Open Order", "list1.ckl", "gui2", 1)，并将语句 edit_set ("Edit_1", "3")修改为 edit_set ("Edit_1", "6")，如图 9-51 所示。

(3) 在 WinRunner 工具栏上单击按钮 From Top 运行该测试文件，测试报错，并弹出图 9-52 所示的报错对话框。

```
# Open Order
    set_window ("Open Order", 1);
    button_set ("Order No.", ON);
    set_window ("Open Order", 5);
    edit_set ("Edit_1", "6");
    win_check_gui("Open Order", "list1.ckl", "gui3", 1);
```

图 9-51　修改后的脚本(2)

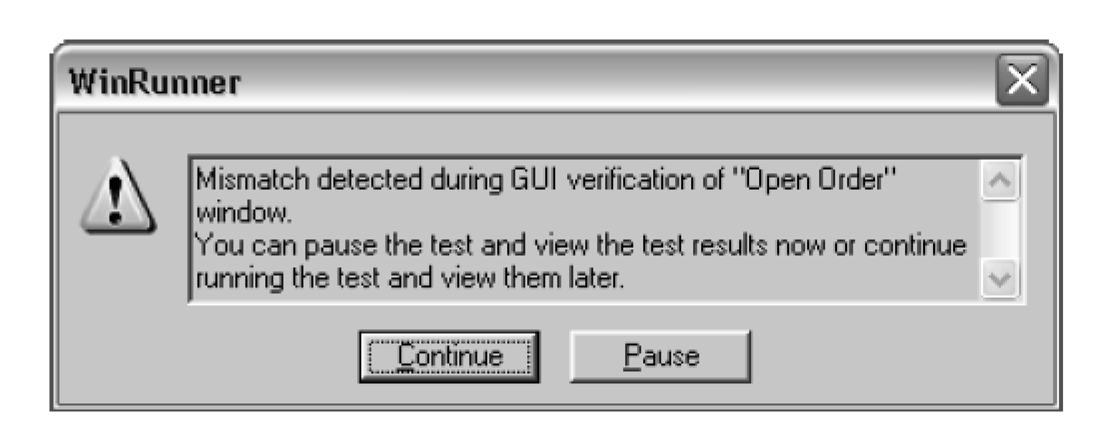


图 9-52　测试报错

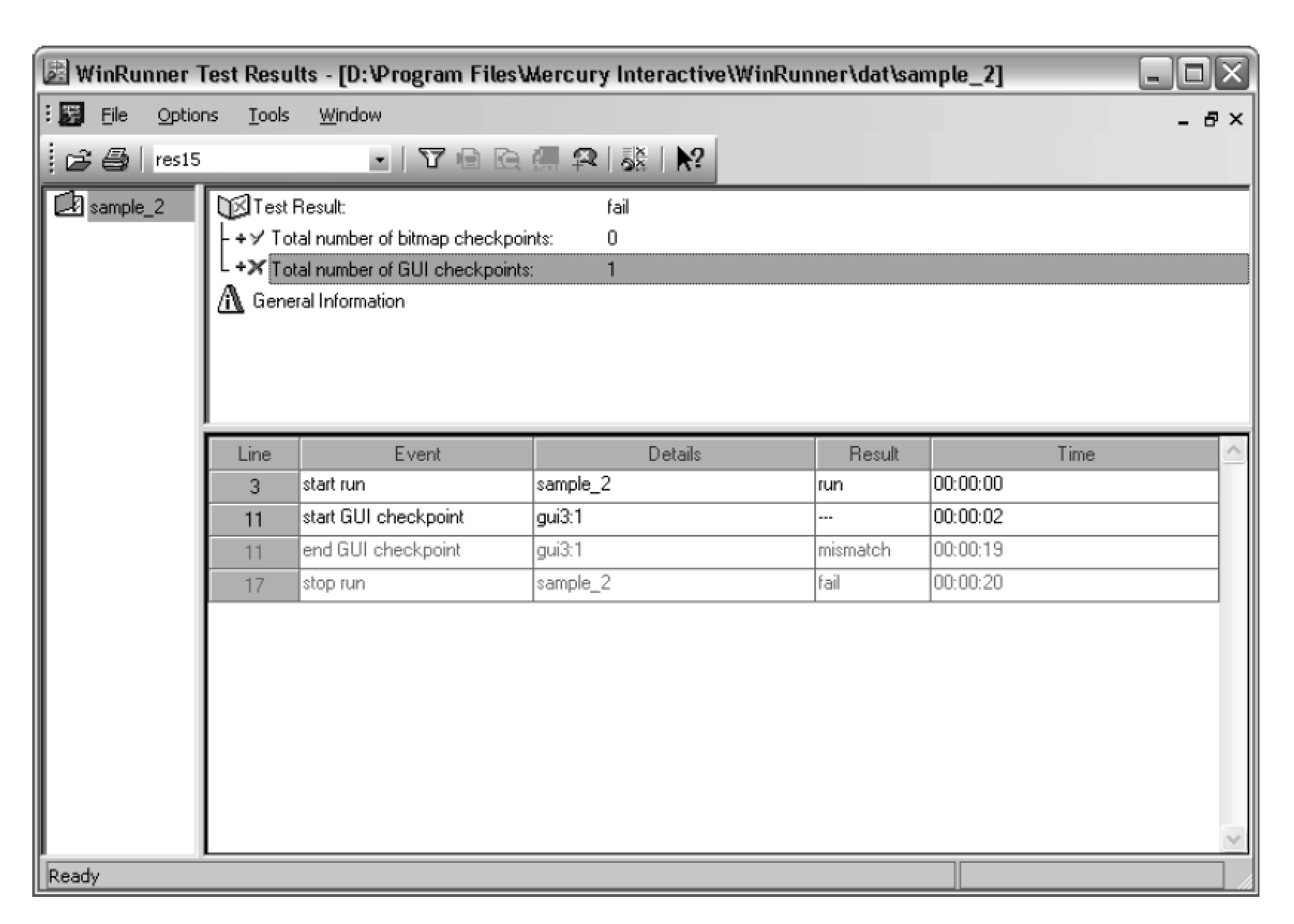


图 9-53　测试结果

(4) 单击 Continue 按钮,完成测试,测试结果显示如图 9-53 所示。

在测试结果窗口的信息列表中,双击红色显示的第 12 行错误信息,弹出 GUI Checkpoint Results 对话框,如图 9-54 所示,显示出产生错误的原因是由于编辑框 Edit_1 的实际值超出了所允许的范围 1～5。

4. 在不同版本的应用程序下的测试

在这一部分,运行与应用程序 Flight 4A 稍有差异的另一版本 Flight 4B,用于模拟在软件开发过程中软件修改之后,如何使用 WinRunner 进行测试来检验软件的变化。

(1) 先运行 Flight 4A。

(2) 创建新的测试文件 sample_3。

(3) 打开 GUI Map 文件(方法如前所述)。

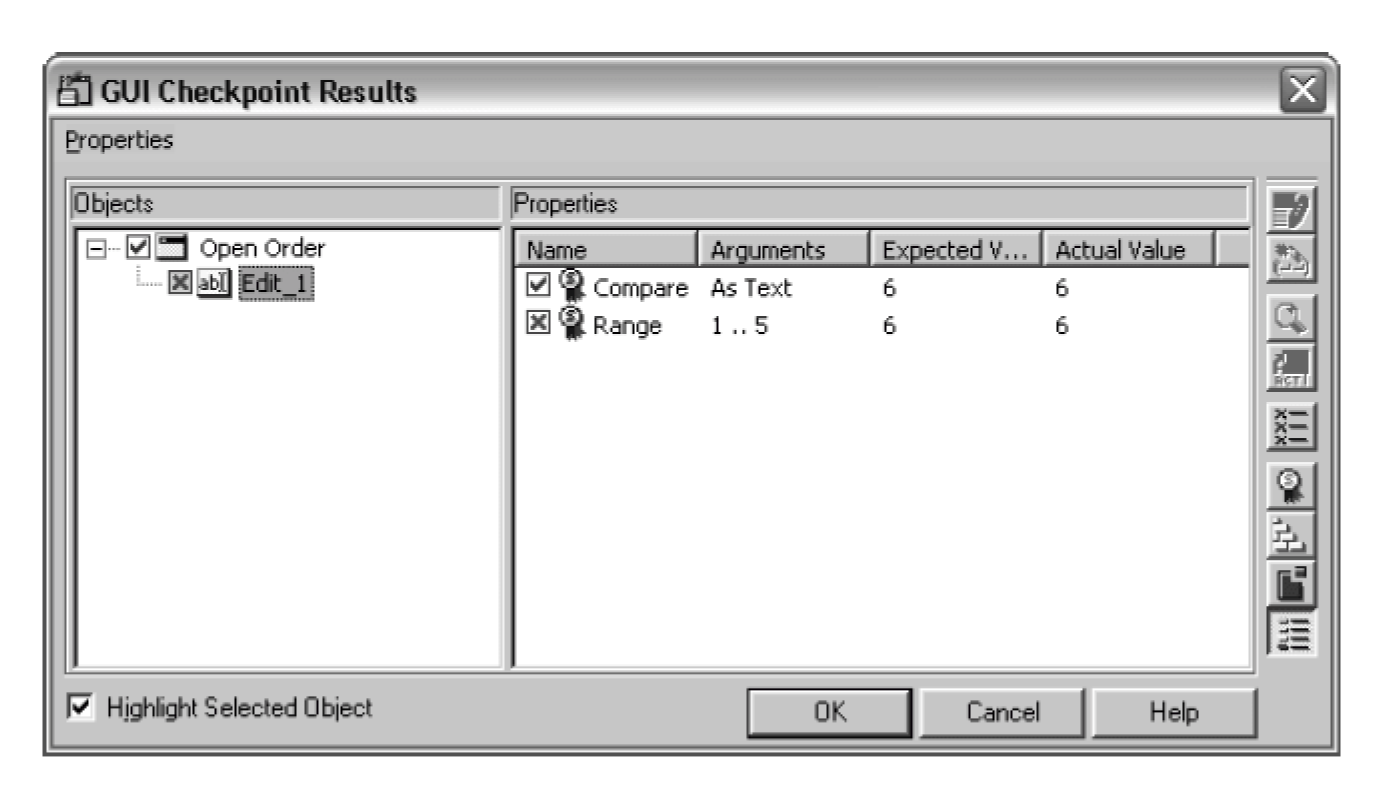


图 9-54 检查点错误信息显示

(4) 单击 WinRunner 菜单栏 Test 中的 Record-Context Sensitive 选项,开始以"上下文相关"(Context Sensitive)模式录制脚本。

(5) 返回 Flight Reservation 窗口,单击其菜单栏 File 中的 Open Order 选项,弹出 Open Order 对话框;在对话框的 Order No. 文本框中输入数字 3,表示打开 3 号订单,单击 OK 按钮,出现 Flight Reservation 窗口(图 9-41);此时,再单击菜单栏 File 中的 Fax Order 选项,弹出 Fax Order No. 3 窗口,如图 9-55 所示。



图 9-55 Fax Order No. 3 窗口

(6) 单击 WinRunner 菜单栏 Insert 中 GUI Checkpoint 级联菜单的 For Multiple Objects 选项,启动 Create GUI Checkpoint 对话框;单击 Add 按钮,出现手形状光标,然后将手形状分别单击 Fax Order No. 3 窗口中的 Class,# Tickets,Ticket Price,From 和 To 文本框,如图 9-56 所示。

(7) 单击鼠标右键,关闭对象捕捉,返回 Create GUI Checkpoint 对话框;单击 OK 按钮,关闭对话框。

单击 WinRunner 工具栏上的"停止"按钮 ■ Stop ,停止录制。此时 WinRunner 脚本窗口出现设置检查点语句,如图 9-57 所示。

(8) 保存测试文件。在工具栏上单击按钮 From Top 运行该测试文件,测试顺利通过。



图 9-56　捕捉窗口中的对象

```
# Fax Order No. 3
    win_check_gui("Fax Order No. 3", "list1.ckl", "gui1", 5);
    win_activate ("Fax Order No. 3");
    set_window ("Fax Order No. 3", 7);
    button_press ("Cancel");
```

图 9-57　设置检查点脚本

(9) 关闭 Flight 4A,运行 Flight 4B。

(10) 在 WinRunner 工具栏上单击按钮 From Top 运行该测试文件,测试报错,并弹出报错对话框,如图 9-58 所示。

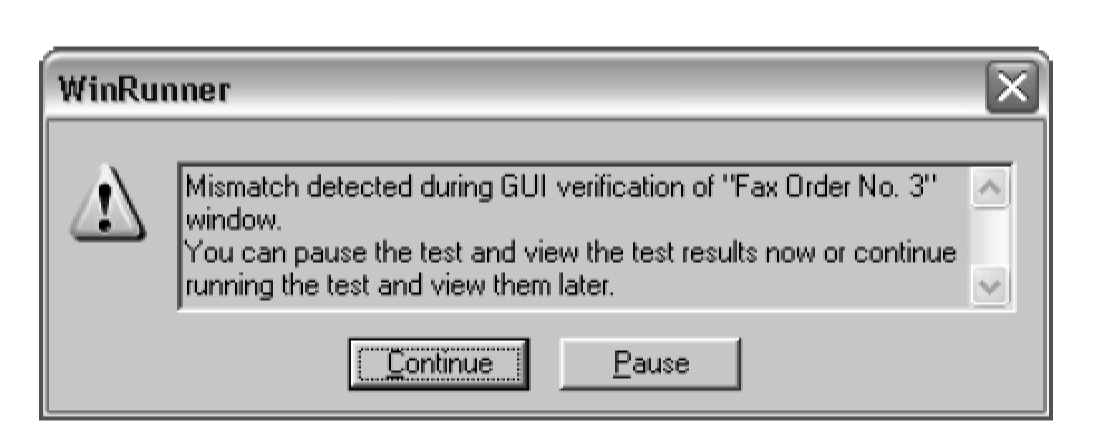


图 9-58　显示测试报错信息

(11) 单击 Continue 按钮,完成测试。测试结果显示如图 9-59 所示。

在测试结果窗口的信息列表中,双击红色显示的第 18 行错误信息,弹出如图 9-60 所示的 GUI Checkpoint Results 对话框,显示出产生错误的原因有两处。

9.6.6　文本检查点

使用 WinRunner 提供的文本检查点功能,可以获取程序界面中的图像或 GUI 对象上的文字,然后再通过手工编写测试脚本的方法来检查文字是否正确。

使用文本检查点,主要可以完成如下功能:

- 验证某个值是否在一定范围内;
- 计算数值是否正确。

文字检查点的建立过程是,当用户指定要读取文字的区域、对象或窗口时,WinRunner 会用 win_get_text 或 obj_get_text 函数读取文字,并将读取到的文字储存到

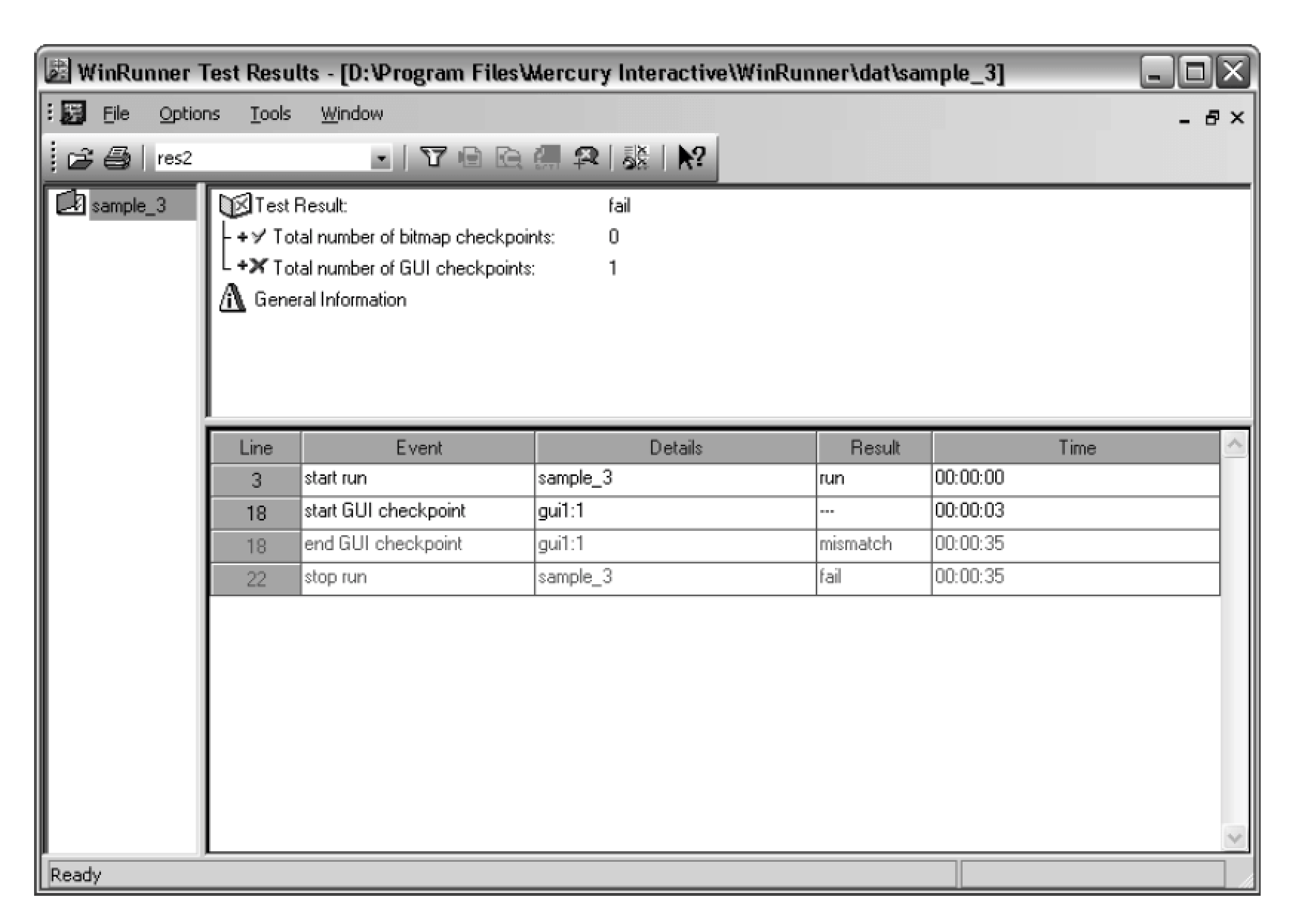


图 9-59　测试结果显示

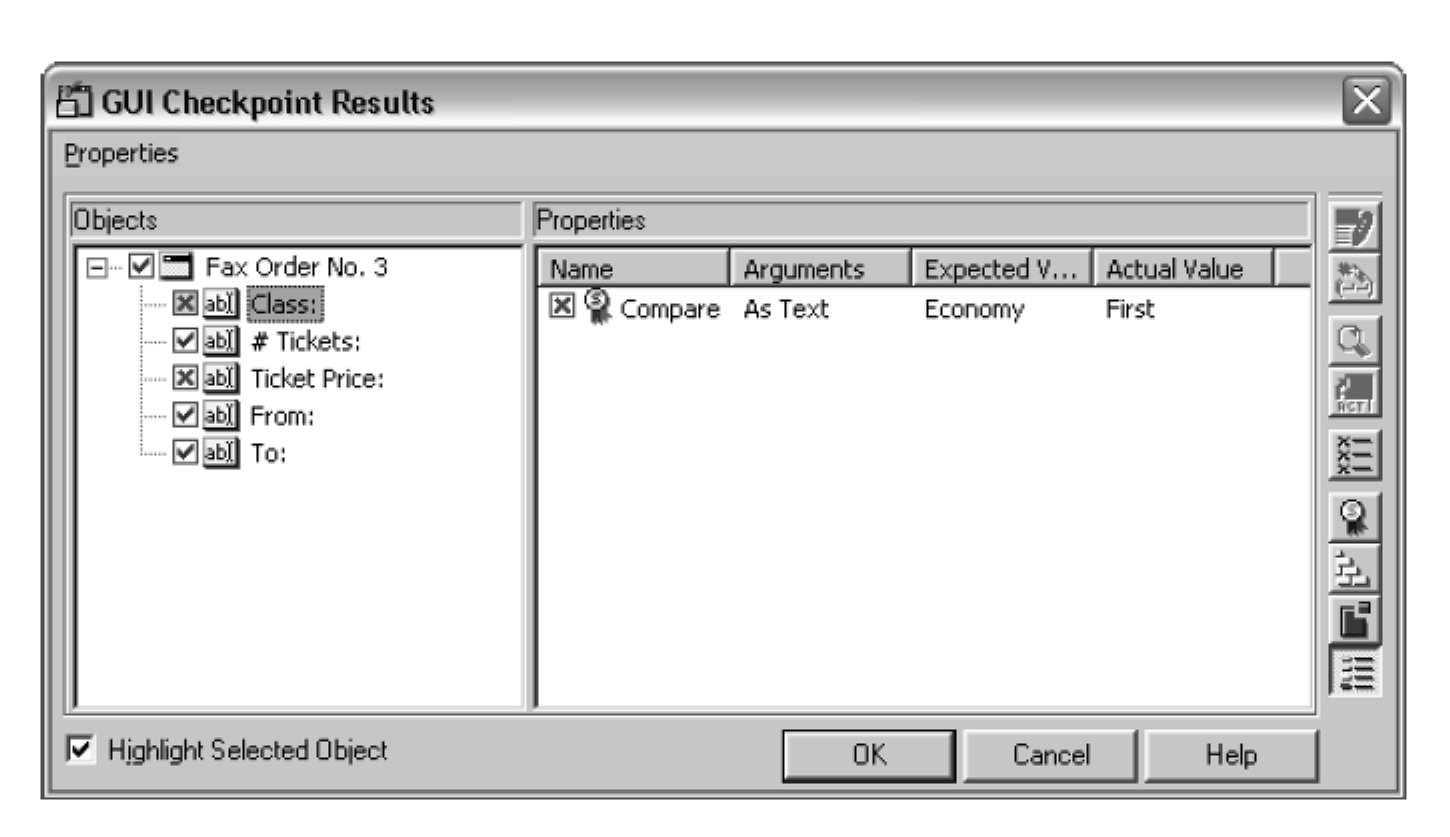


图 9-60　显示检查点错误信息

变量中，然后再以手工编写测试脚本方式，检查变量中的文字是否为预期的文字。注意，如果是验证标准的 GUI 对象如按钮、功能选单等的文字，建议用 GUI 检查点，以免去因手动编写测试脚本带来的不便。

下面用一个实例演示如何使用文字检查点，该示例的主要功能是检查“记事本”文件内容的变化。

(1) 启动 WinRunner。

(2) 创建新的测试文件 sample_4。

(3) 打开 Windows 的“记事本”窗口，填入如图 9-61 所示的数据，并保存为 recorder.txt。

(4) 单击菜单栏 Test 中的 Record-Context Sensitive 选项，开始以“上下文相关”(Context Sensitive)模式录制一个脚本。

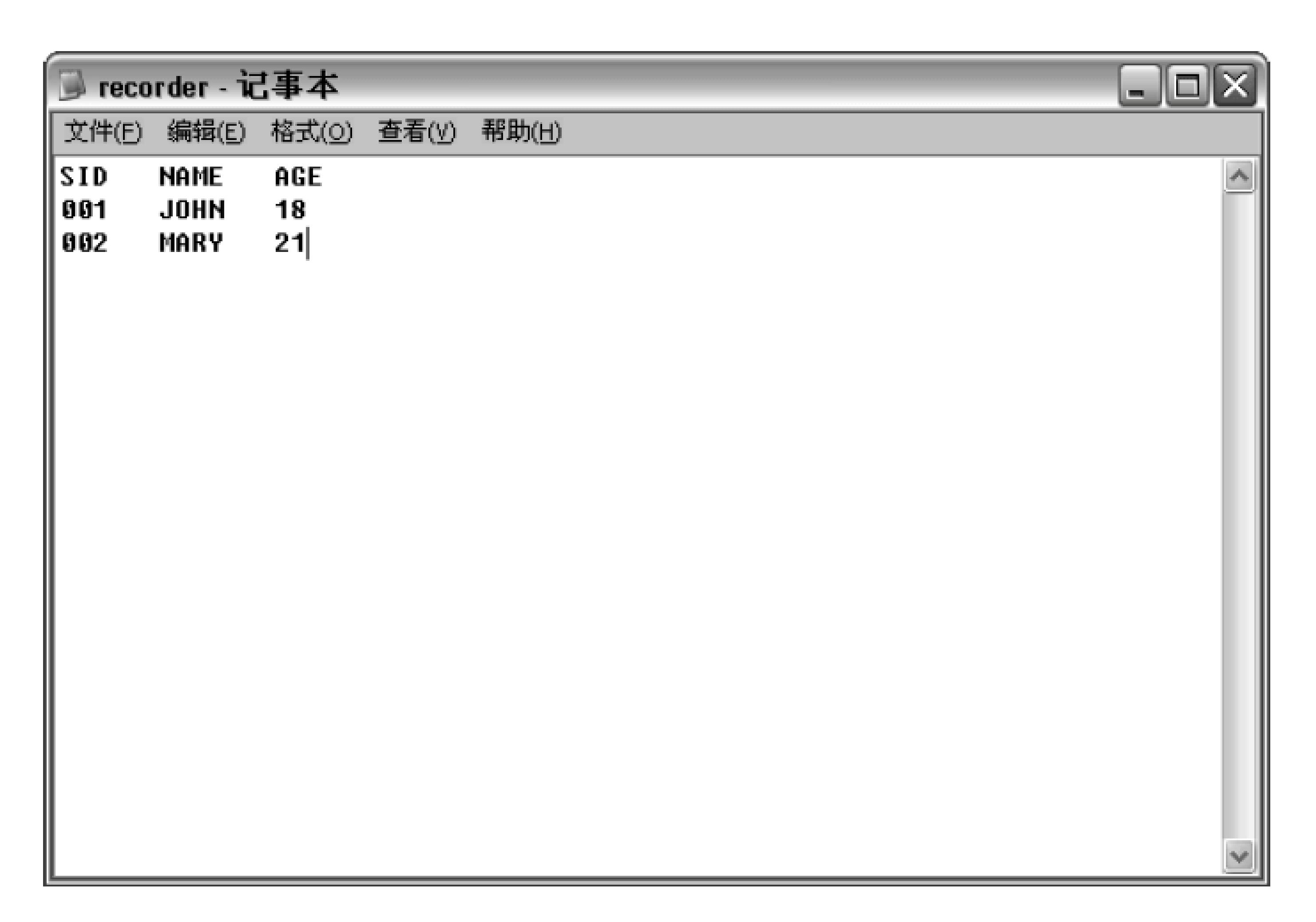


图 9-61　recorder. txt 文件内容

(5) 单击 WinRunner 菜单栏 Insert 中 Get Text 级联菜单的 For Object/Window 选项。此时，出现手形状光标，将手形状指向 recorder“记事本”窗口的编辑区，如图 9-62 所示，单击鼠标右键，停止捕捉，回到 WinRunner 主界面。

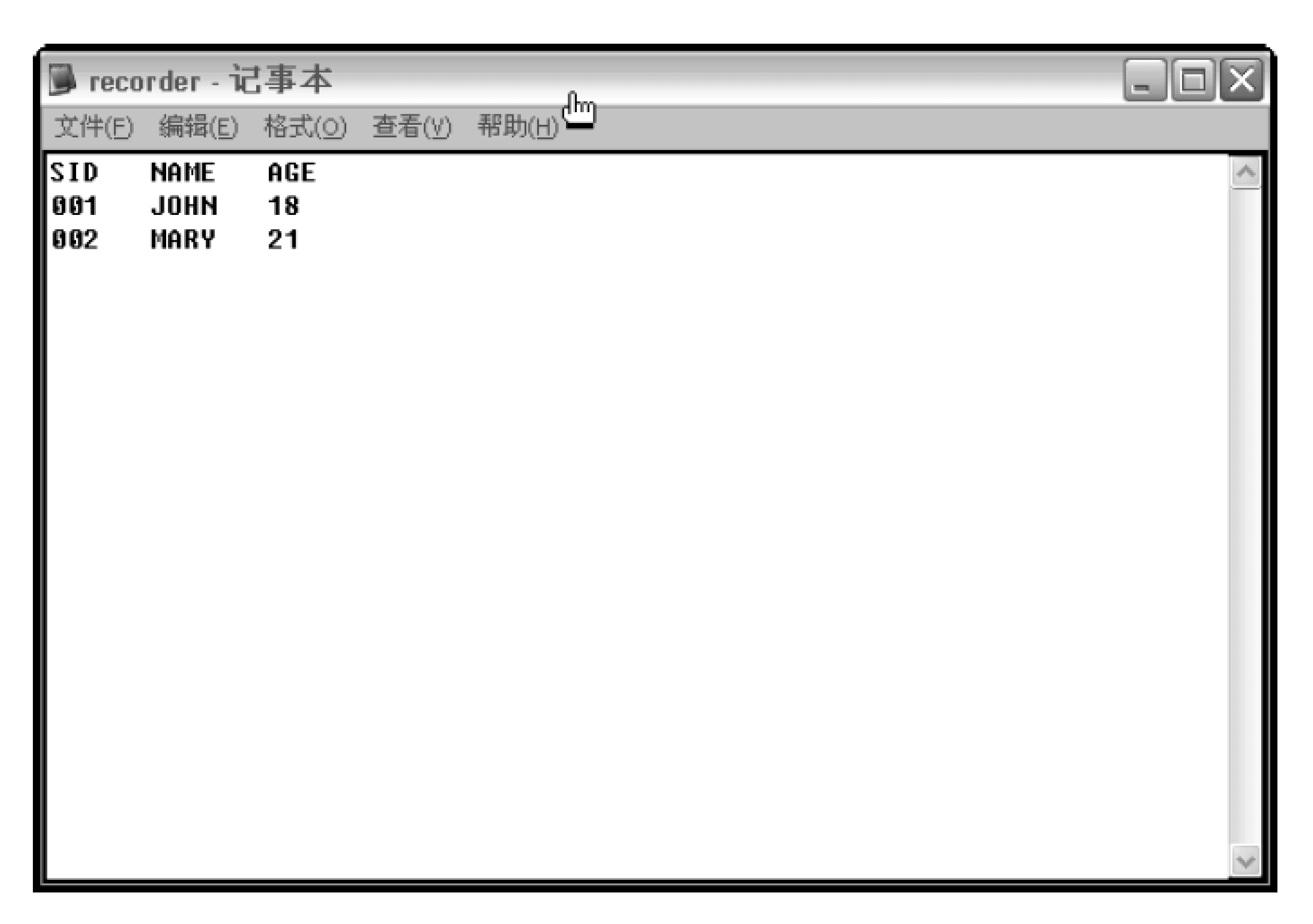


图 9-62　修改后的 recorder. txt

(6) 单击 WinRunner 工具栏上的“停止”按钮 ■ Stop，停止录制。此时 WinRunner 脚本窗口出现检查点语句 obj_get_text，如图 9-63 所示。其中前 3 条注释语句是 WinRunner 捕捉到的文本信息，该信息保存在 obj_get_text 函数的参数 text 变量中。

```
    # SID   NAME   AGE
    # 001   JOHN   18
    # 002   MARY   21
# recorder - 记事本
    set_window("recorder - 记事本", 5);
    obj_get_text("Edit", text);
```

图 9-63　检查点脚本语句

(7) 为了设置文字检查点，在脚本语句 obj_get_text 之后添加条件判断语句，如图 9-64 所示。如果 text 变量的信息与 if 条件匹配，则在测试结果窗口返回通过信息“Text is correct!”，否则返回失败信息。

```
    # SID    NAME    AGE
    # 001    JOHN    18
    # 002    MARY    21

# recorder - 记事本
    set_window("recorder - 记事本", 5);
    obj_get_text("Edit", text);
#////////////////////////////////////////////////////////////////////
    if (text=="SID    NAME    AGE\r\n001    JOHN    18\r\n002    MARY    21")
        tl_step("Notepad Text",0,"Text is correct!");
    else
         tl_step("Notepad Text",1,"Text is not correct!");
```

图 9-64　返回通过或失败信息脚本

(8) 在工具栏上单击按钮 From Top 运行该测试文件,测试顺利通过,并显示如图 9-65 所示的测试结果。

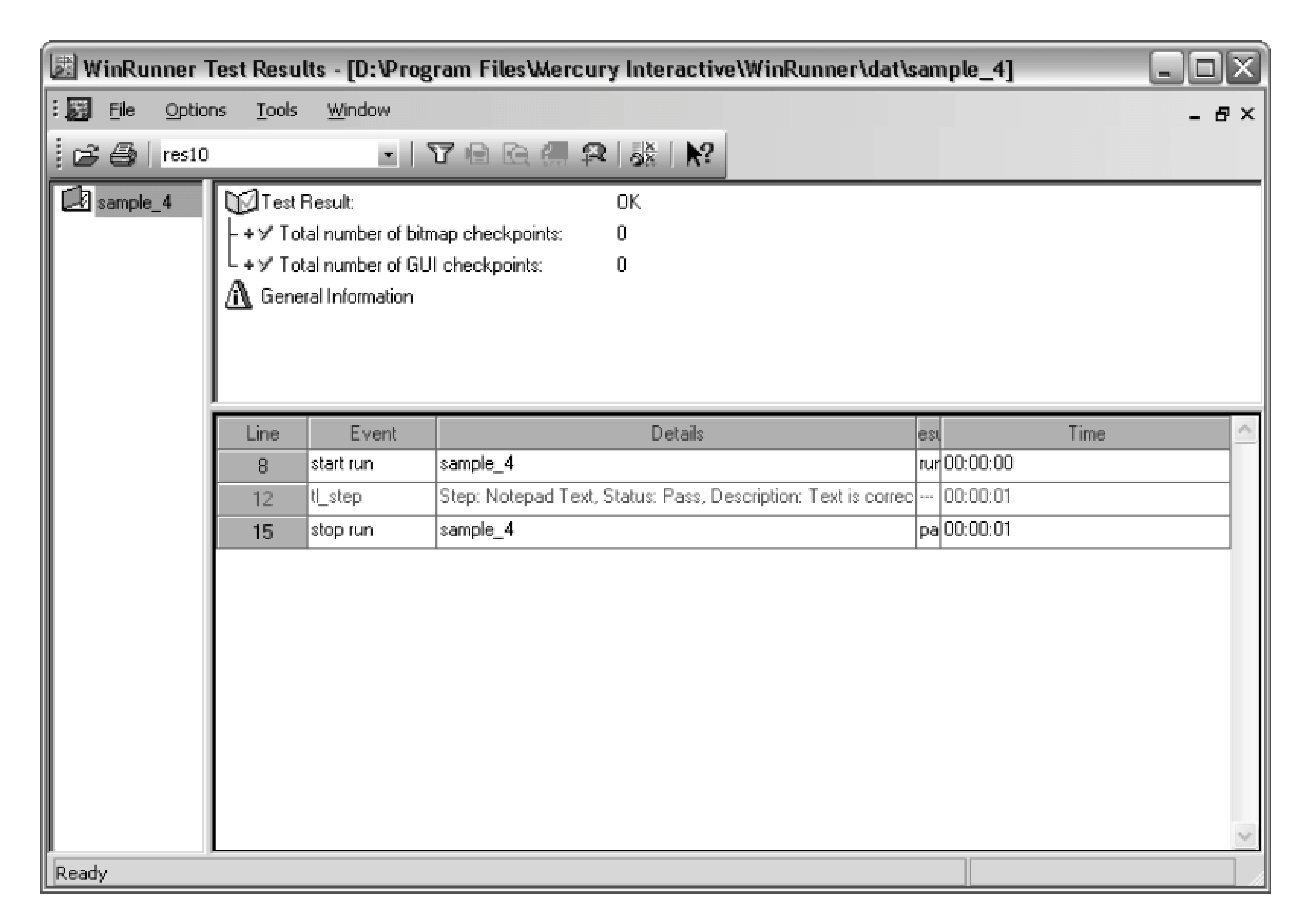


图 9-65　测试结果显示

(9) 修改 recorder 记事本文件,将 002 号记录的 AGE 改为 20,保存文件。然后再运行 sample_4 测试文件,测试报错,并显示如图 9-66 所示的测试结果。

9.6.7　图像检查点

针对应用程序中的图像,WinRunner 提供了图像检查点(Bitmap Checkpoint),它是以像素点(pixel)为单位比较图像的。

WinRunner 使用以下 3 种方式建立图像检查点:

(1) 屏幕区域:以鼠标拖曳的方式确定图像检查点的区域。

(2) 窗口:以整个窗口作为图像检查点的区域。

(3) GUI 对象:以整个 GUI 对象作为图像检查点的区域。

WinRunner 能够直接取得方框区域部分并储存成预期值,然后在测试脚本中插入 obj_check_bitmap 或 win_check_bitmap 函数(用于判断所建立的图像检查点是针对区域、GUI 对象还是窗口对象)。运行测试脚本时,WinRunner 会比对运行时的图像与预期

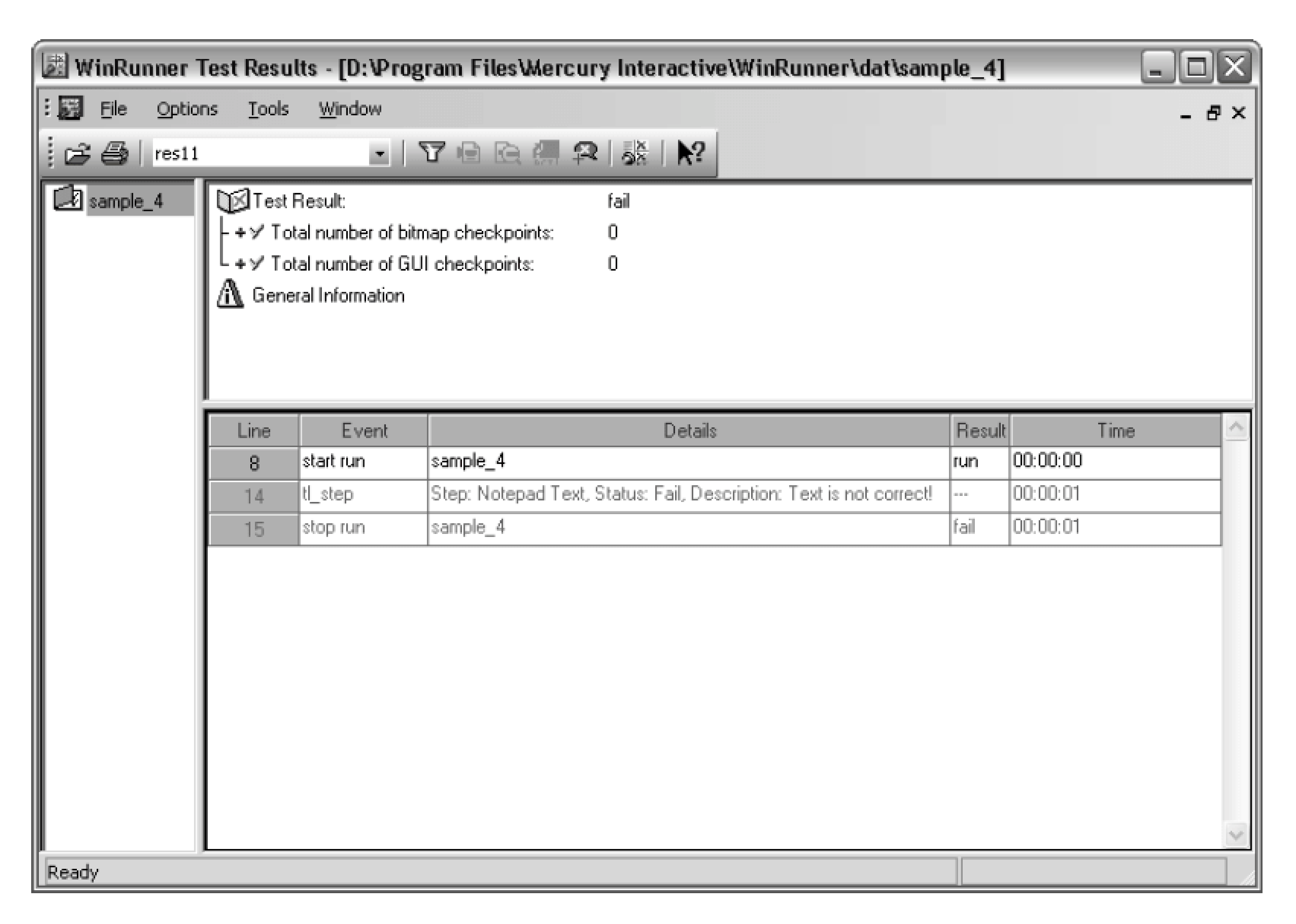


图 9-66　修改后的测试结果显示

的图像,并将结果显示在测试结果窗口中;如果存在不一致,系统就会报错,并在测试结果窗口显示出错信息。

下面用一个实例演示如何使用图像检查点。

(1) 启动 WinRunner 和 Flight 4A。

(2) 创建新的测试文件 sample_5。

(3) 打开 GUI Map 文件。选择 WinRunner 菜单栏 Tools 中的 GUI Map Editor 选项,启动 GUI Map Editor 窗口;选择 GUI Map Editor 菜单栏 Files 中的 Open 选项,打开 flight4a. gui 文件。

(4) 单击菜单栏 Test 中的 Record-Contex Sensitive 选项,开始以"上下文相关"(Context Sensitive)模式录制一个脚本。

(5) 在 Flight Reservation 窗口中,单击 File 菜单中的 Open Order 选项,弹出 Open Order 对话框;在 Order No. 文本框中输入数字 1,表示打开 1 号订单;单击 OK 按钮。

(6) 单击 File 菜单中的 Fax Order 选项,出现 Fax Order 窗口。在 Fax Number 文本框中输入 10 位数的传真号码(可以是任意数字),如图 9-67 所示。

(7) 移动 Fax Order 窗口。

(8) 按 F2 功能键,或者单击工具列上的按钮 Record ,可以将录制模式从 Context Sensitive Mode 切换到 Analog Mode。

(9) 在 Agent Signature 编辑框中签上用户的名字(使用鼠标画图),如图 9-68 所示。

(10) 按 F2 功能键,或者单击工具栏上的"录音"按钮 Record ,可以将录制模式从 Analog Mode 切换到 Context Sensitive Mode。

(11) 单击 WinRunner 菜单栏 Insert 中 Bitmap Checkpoint 级联菜单的 For Object/Window 选项;使用手形状鼠标指向 Agent Signature 签名窗口后单击,如图 9-69 所示;此



图 9-67 输入传真号

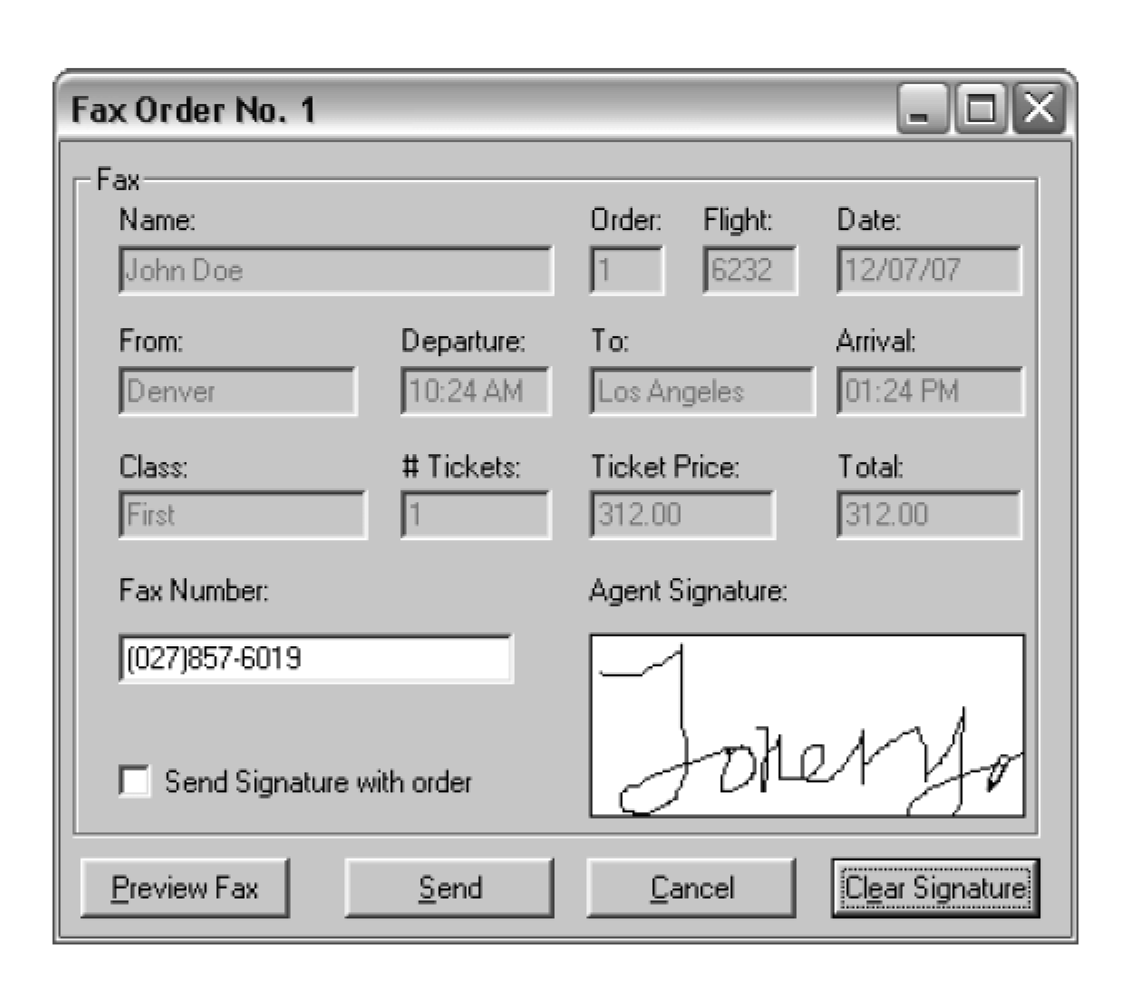


图 9-68 签名



图 9-69 捕捉签名窗口

时，WinRunner 将捕获一个 Bitmap，并在脚本中插入 Obj_Check_Bitmap 声明语句。

（12）单击 Clear Signature 按钮，清除签名。

（13）为清除签名后的签名窗口创建一个 Bitmap Checkpoint，同步骤(11)。

（14）在 Fax Order No. 1 窗口中单击 Cancel 按钮，关闭窗口。

（15）停止录制，保存 sample_5。保存临时 GUI 文件到 Global GUI Map 文件 flight4a.gui。系统生成的有关图像检查点脚本语句，如图 9-70 所示。

```
    obj_check_bitmap("(static)", "Img6", 7);

# Fax Order No. 1
    win_activate ("Fax Order No. 1");
    set_window ("Fax Order No. 1", 2);
    button_press ("Clear Signature");
    obj_check_bitmap("(static)", "Img7", 3);
```

图 9-70　图像检查点脚本

（16）在工具栏上单击按钮 From Top 运行该测试文件，测试顺利通过，并显示如图 9-71 所示的测试结果。

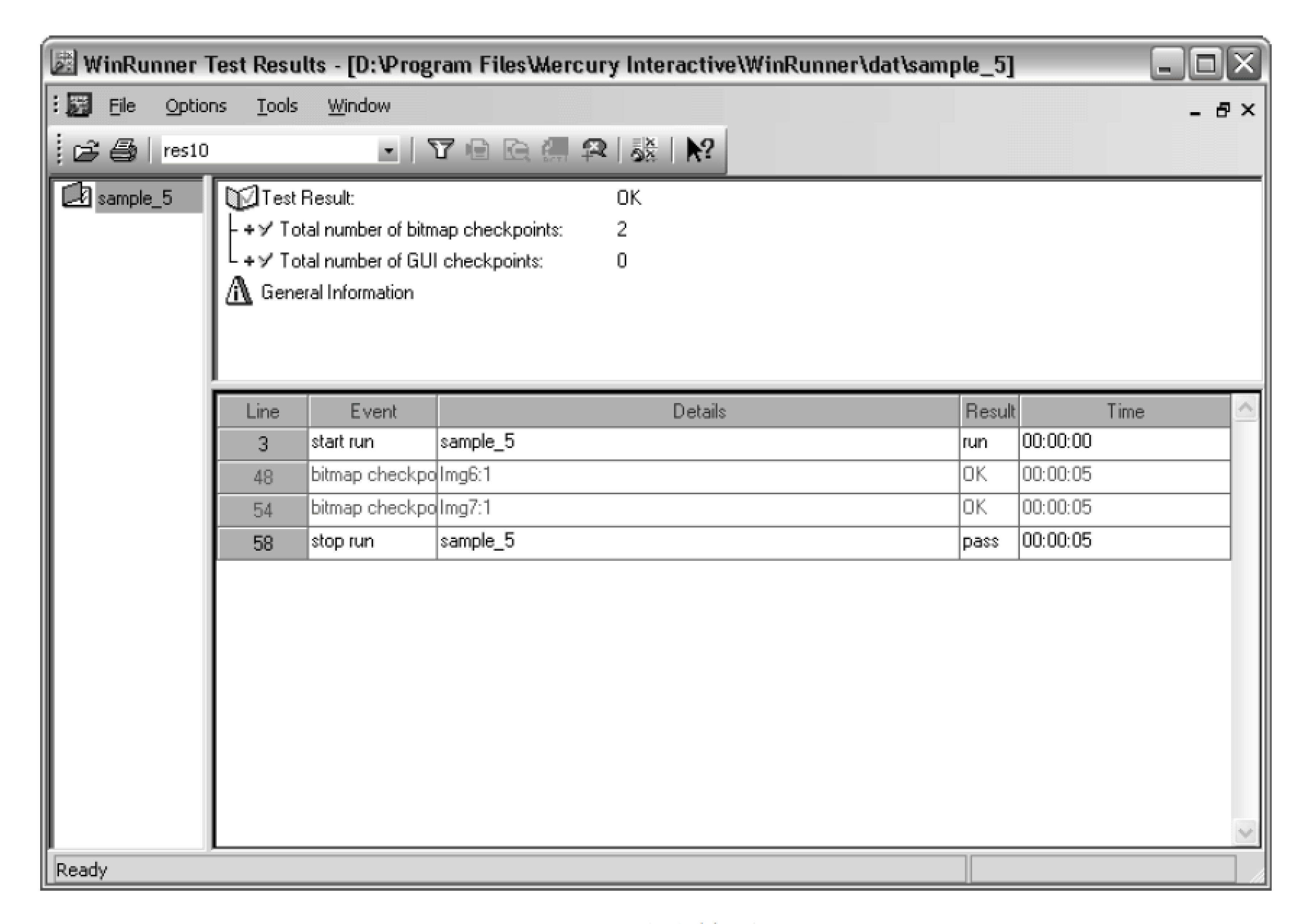


图 9-71　测试结果显示

（17）关闭 Flight 4A，运行 Flight 4B。

（18）在 WinRunner 工具栏上单击按钮 From Top 运行该测试文件，测试报错，并弹出报错对话框，如图 9-72 所示。

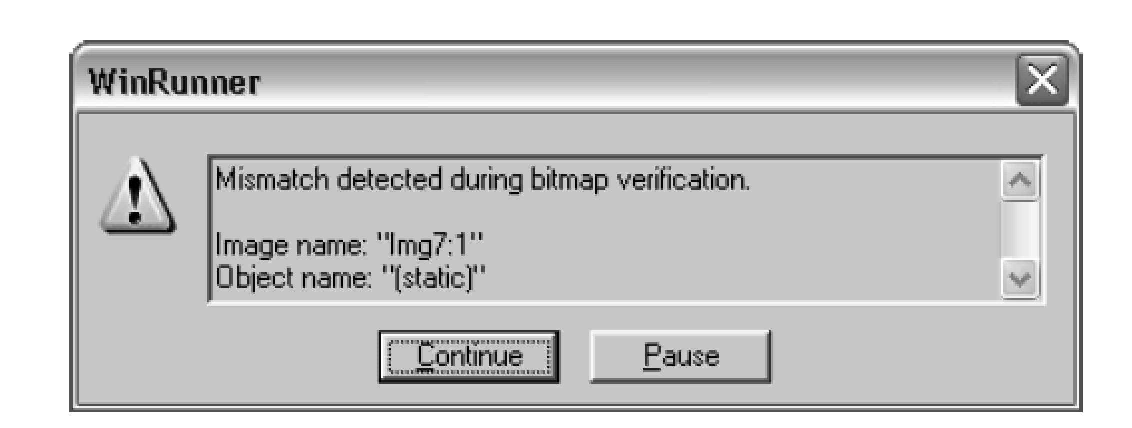


图 9-72　报错信息显示

（19）单击 Continue 按钮，完成测试，测试结果显示如图 9-73 所示。

（20）在测试结果窗口的信息列表中，双击红色显示的第 54 行错误信息，弹出 Bitmap

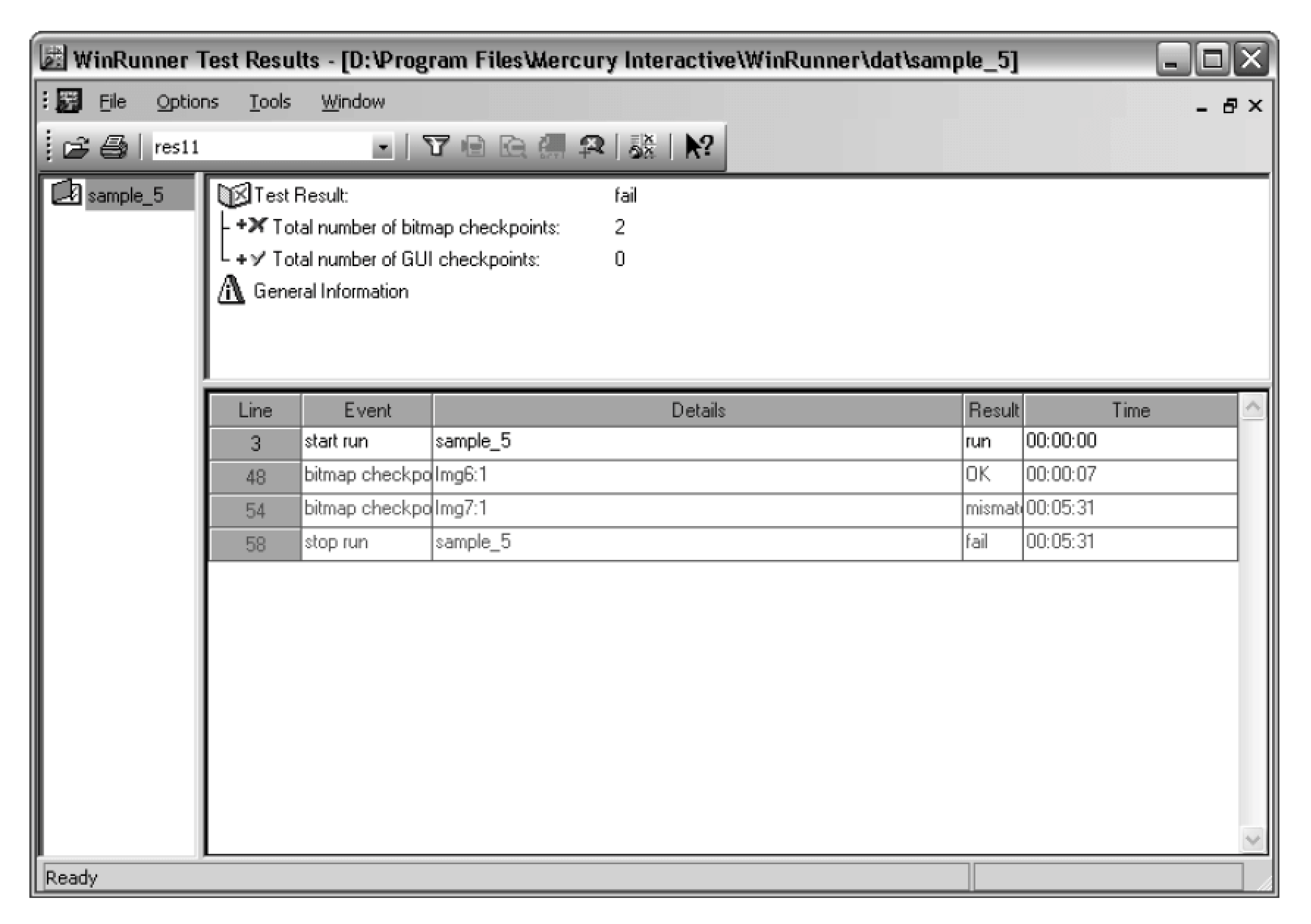


图 9-73　显示测试结果



图 9-74　Bitmap 对比

窗口，如图 9-74 所示。

Bitmap 窗口清楚地显示了产生错误的原因是由于 img7. bmp 的期望值与实际值不匹配。这是由于 Flight 4B 程序的 Fax Order 窗口的 Clear Signature 按钮(参看图 9-68)是失效的，所以无法将前面的签名清除。需要说明的是，这是系统有意设计的错误，主要是为了模拟软件开发中的一些错误。

9.6.8　使用函数生成器

WinRunner 提供了强大的函数生成器来帮助用户编写测试脚本。在本节，将通过一个简单的例子来学习如何使用函数生成器来编写测试脚本。

该示例使用的测试程序仍然是 Flight 4A，主要功能是验证 Flight 4A 程序的 Fax Order 窗口中的票数、票价和总额字段是否正确。

(1) 启动 WinRunner 和 Flight 4A。

(2) 创建新的测试文件 sample_6。

(3) 打开 GUI Map 文件。选择 WinRunner 菜单栏 Tools 中的 GUI Map Editor 选项，启动 GUI Map Editor 窗口；选择 Files 菜单中的 Open 选项，打开 flight4a. gui 文件。

(4) 单击 Test 菜单中的 Record-Context Sensitive 选项，开始以“上下文相关”

(Context Sensitive)模式录制一个脚本。

(5) 单击 Flight Reservation 窗口 File 菜单中的 Open Order 选项，弹出 Open Order 对话框；在 Order No. 文本框中输入数字 5，表示打开 5 号订单；单击 OK 按钮。

(6) 单击 File 菜单中的 Fax Order 选项，打开 Fax Order No. 5 窗口，如图 9-75 所示。

(7) 单击 Cancel 按钮，关闭 Fax Order No. 5 窗口；再单击 WinRunner 工具栏上的"停止"按钮 ■ Stop，停止录制；保存 sample_6，保存临时 GUI 文件到 Global GUI Map 文件 flight4a.gui。脚本文件如图 9-76 所示。

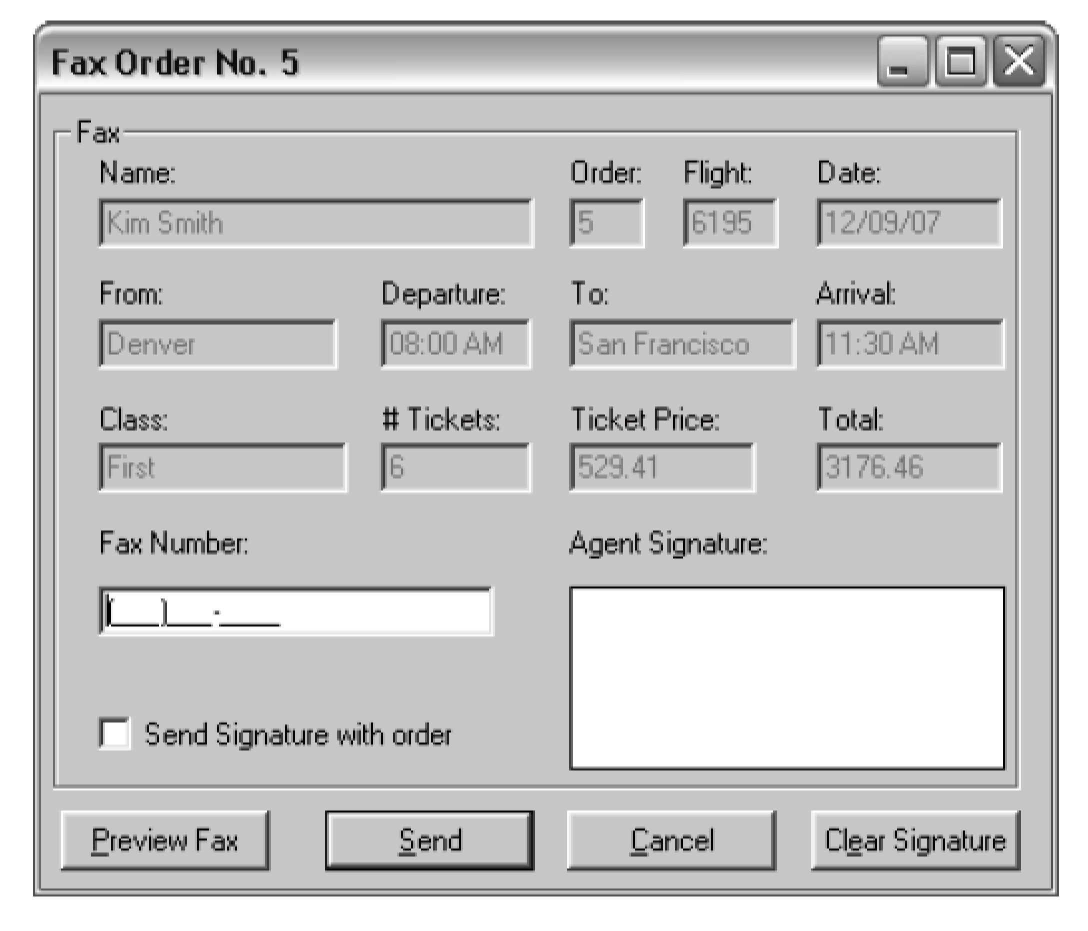
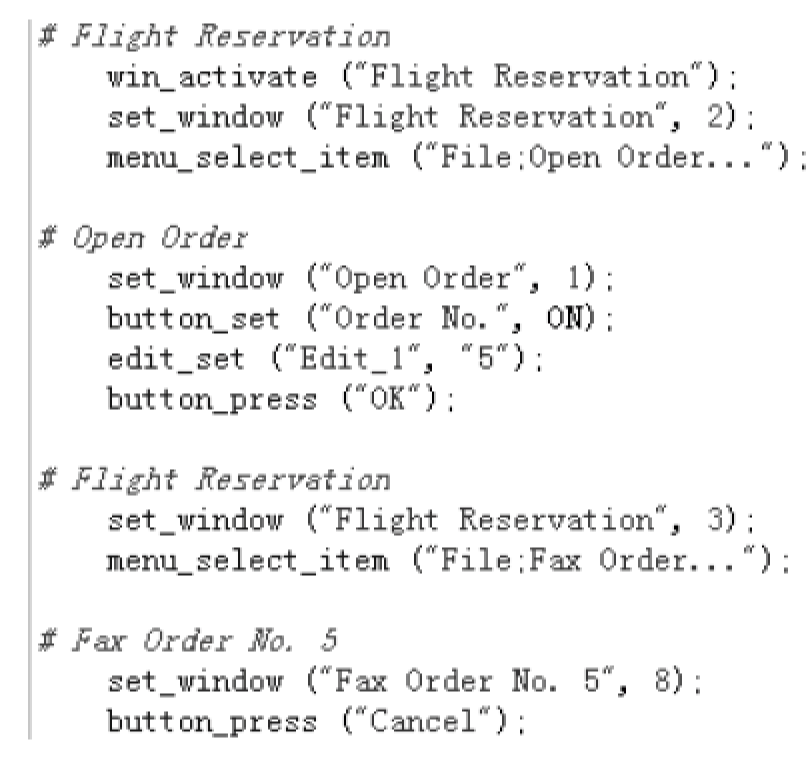


图 9-75　Fax Order No. 5 窗口



```
# Flight Reservation
    win_activate ("Flight Reservation");
    set_window ("Flight Reservation", 2);
    menu_select_item ("File;Open Order...");

# Open Order
    set_window ("Open Order", 1);
    button_set ("Order No.", ON);
    edit_set ("Edit_1", "5");
    button_press ("OK");

# Flight Reservation
    set_window ("Flight Reservation", 3);
    menu_select_item ("File;Fax Order...");

# Fax Order No. 5
    set_window ("Fax Order No. 5", 8);
    button_press ("Cancel");
```

图 9-76　脚本文件内容

(8) 在脚本的最后一行 button_press("Cancel")的上方插入一空行。重新在 Flight Reservation 窗口中打开 Fax Order No. 5 窗口(图 9-75)。

(9) 选择 WinRunner 菜单栏 Insert 中的 Function，单击 For Object/Window 选项，出现手形状光标。使用手形状指针指向 Fax Order No. 5 窗口中的票数字段 # Tickets，如图 9-77 所示。



图 9-77　捕捉 # Tickets 文本框

(10) 单击票数字段#Tickets,打开函数生成器窗口,如图 9-78 所示;修改 edit_get_text 函数中默认的参数 text 为 tickets,然后单击 Paste 按钮。

(11) 同前面的步骤,分别单击 Ticket Price 字段,添加票价查询字段,修改默认变量名 text 为 price;单击 Total 字段,添加票价总额查询字段,修改默认变量名 text 改为 total,脚本文件如图 9-79 所示。

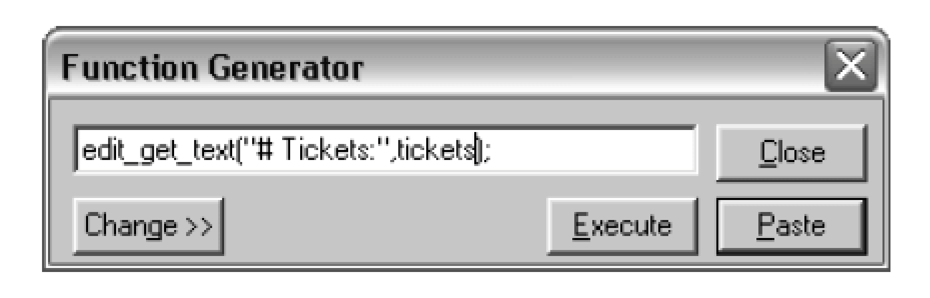


图 9-78　函数生成器窗口

```
# Fax Order No. 5
    set_window ("Fax Order No. 5", 8);
    edit_get_text("# Tickets:",tickets);
    edit_get_text("Ticket Price:",price);
    edit_get_text("Total:",total);

    button_press ("Cancel");
```

图 9-79　添加字段脚本

(12) 在脚本 edit_get_text("Total: ",total) 之后插入一条件语句,如图 9-80 所示。该条件语句用来验证票价总额文本框中的值是否等于票价文本框的值乘以票数文本框的值。如果相符,则通过测试,否则报错。

```
# Fax Order No. 5
    set_window ("Fax Order No. 5", 8);
    edit_get_text("# Tickets:",tickets);
    edit_get_text("Ticket Price:",price);
    edit_get_text("Total:",total);
    if (total==tickets*price)
        tl_step("Ticket verification",0,"total is correct!");
    else
        tl_step("Ticket verification",1,"total is not correct!");
```

图 9-80　添加条件语句脚本

(13) 在工具栏上单击按钮 From Top 运行该测试文件,测试顺利通过,并显示如图 9-81 所示的测试结果。

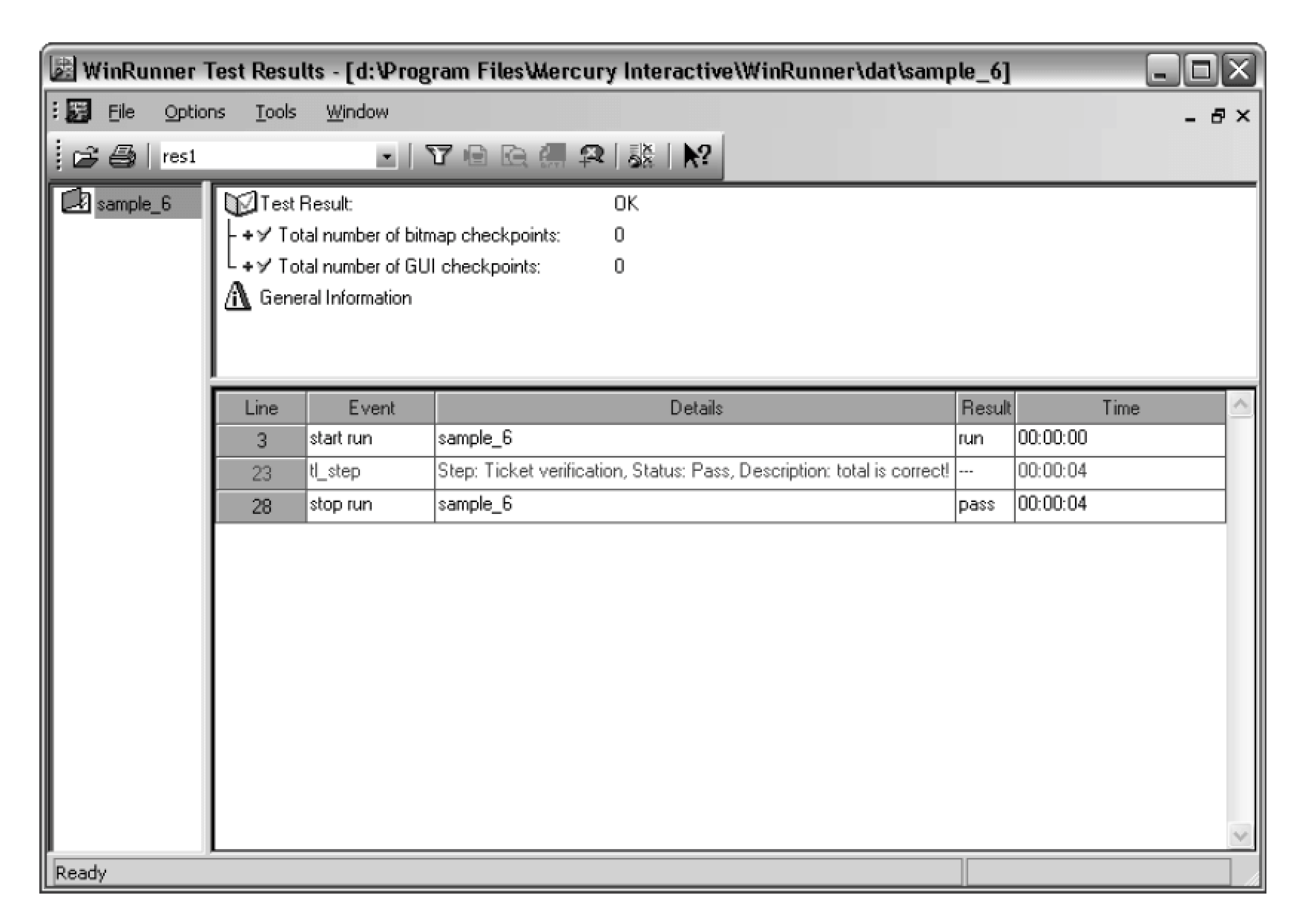


图 9-81　测试结果窗口

9.6.9　数据驱动测试

在 WinRunner 中，可以将测试脚本转换成数据驱动测试脚本，并建立一个数据表，用来提供测试所需的多组数据。这样做的目的是可以用多组不同的数据去执行测试脚本。

当执行数据驱动测试脚本时，WinRunner 会读取数据表的每一组数据，然后分别执行。每执行一次称为一次迭代(Iteration)，数据表有几组数据，WinRunner 就会进行几次迭代。WinRunner 会在最后的测试结果窗口中显示每一次迭代的测试结果。

由于篇幅所限，不能对数据驱动测试进行更深入的介绍，感兴趣的读者可以参考其他相关的书籍。

9.7　小结

本章介绍了自动化测试的必要性、优点、认识误区、原理和实施流程等，介绍了主流的测试工具，还对著名的测试工具提供商 IBM Rational 和 HP Mercury 公司的自动化测试解决方案进行了介绍，最后较详细地介绍了如何使用 WinRunner 进行功能测试。

对测试工具的了解可以参看各工具提供商的官方网站及其他相关网站的资料。

习　　题

1. 为什么自动化测试是必须且可行的?
2. 简述自动化测试的优点。
3. 企业引进自动化测试后测试工作的效率一定会提高吗? 为什么?
4. 简述自动化测试的实施流程。
5. 什么是脚本? 脚本分为哪几类? 各有何作用?
6. 试谈谈你对 IBM Rational 软件自动化测试解决方案的认识。
7. WinRunner 录制测试的模式分别有哪几种?
8. WinRunner 的测试流程包括哪些步骤?
9. WinRunner 中的 GUI Map File 模式有哪两种? 它们有何区别?
10. 为什么说 WinRunner 能实现回归测试? 谈谈你的观点。
11. 试根据本章介绍的方法使用 WinRunner 对你开发的软件版本进行测试，并谈谈你对 WinRunner 的认识。

第 10 章

软件质量和质量保证

本章要点：

- 软件质量的含义。
- 三个重要的软件质量模型。
- 软件度量的含义和作用。
- 软件度量的三个方面。
- CMM 的 5 个等级的含义。
- CMM 的逻辑结构。
- CMMI 和 CMM 的异同点。
- 软件质量保证的目的和工作内容。
- 软件质量保证与测试的区别。

在第 1 章曾说过，软件测试的目标不仅是尽可能多地发现软件中的错误，还要对软件质量进行度量和评估，以提高软件质量。

为了能够定量地对软件产品进行度量和评估，一系列的软件质量模型被提出了。它们使软件开发企业在提升自己的软件过程能力及开发能力时，有据可依，并能获得指导。

本章介绍了与软件测试密切相关的一些概念或活动：软件质量、软件质量模型、软件度量、软件能力成熟度模型及能力成熟度模型集成、ISO 9000 标准、软件质量保证(SQA)等。

10.1 软件质量

10.1.1 软件质量的含义

首先来看一些关于质量的定义。

- 1970 年，Juran 和 Gryna 把质量定义为“适于使用”。
- 1979 年，Crosby 将质量定义为“符合需求”。
- 在 GB/T 6583-ISO 8404(1994 版)中，将质量定义为“反映实体满足明确和隐含需要的能力的特性的总和”，这里的实体是“可以单独描述和研究的事物”，如产品、

活动、过程、组织的体系等。

- 在 ISO 9000：2000 中，将质量定义为“一组固有特性满足要求的程度”。

至于软件质量，很容易从上述质量的定义进行扩展。IEEE 对软件质量的定义是，软件质量是系统、部件或过程满足顾客或者用户需要或期望的程度，以及系统、部件或过程满足规定需求的程度。

10.1.2　软件质量模型

实际上，不同的人从不同的角度来看软件质量，会有不同的理解。从用户的角度看，质量就是满足客户的需求；从开发者的角度看，质量就是与需求说明保持一致；从产品的角度看，质量就是产品的内在特点；从价值的角度看，质量就是客户是否愿意购买。

因此，为了避免软件质量评价的盲目性，需要制定软件质量模型，从多方面、客观地、定量地对软件质量进行度量和评价。

目前已有很多种软件质量模型，它们分别定义了不同的软件质量属性。比较常见的三个软件质量模型是 McCall 模型、Boehm 模型和 ISO 9126。

McCall 模型是 McCall 等在 1977 年提出的，如图 10-1 所示。

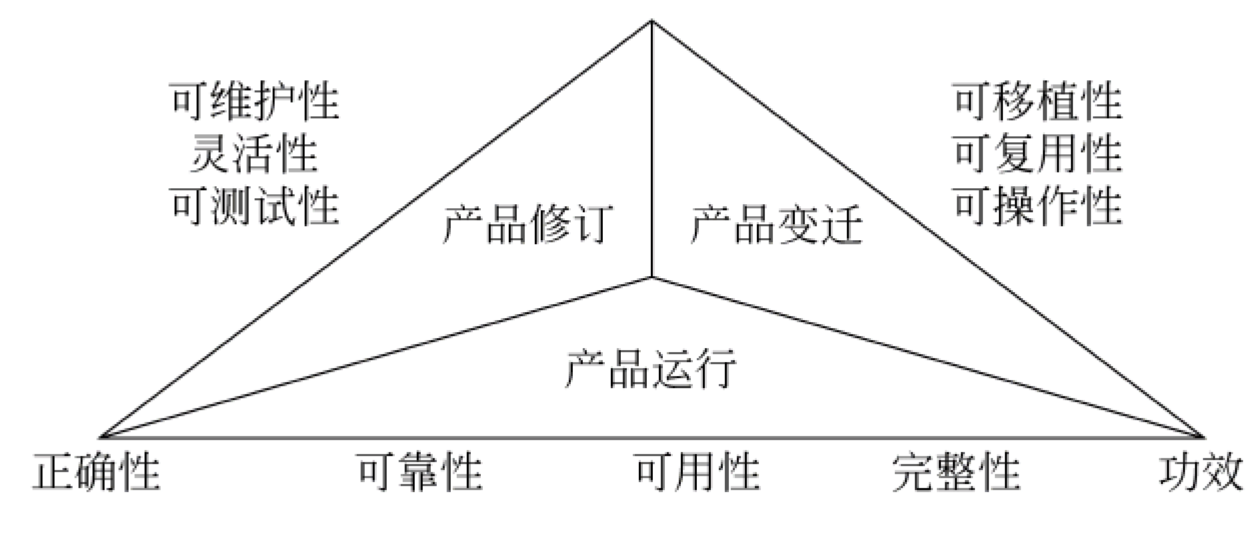


图 10-1　McCall 质量模型

McCall 质量模型将软件质量分为三个重要的方面：操作特性（产品运行）、承受可改变能力（产品修订）和新环境适应能力（产品变迁）。

Boehm 模型是 Boehm 等 1978 年提出的，如图 10-2 所示。

Boehm 质量模型是一个分层的模型，除了包含用户的期望和需要，它还包含了 McCall 模型所没有的硬件质量特性。

Boehm 质量模型关注不同类型的用户需要。第一类用户是初始用户，第二类用户是要将软件移植到其他软硬件系统中使用的用户，第三类用户是系统维护人员。也可以说，Boehm 质量模型反映了不同类型的用户对软件质量的理解。

1991 年，国际标准化组织（International Standards Organization，ISO）颁布了 ISO 9126—1991 标准《软件产品评价—质量特性及其使用指南》。我国也于 1996 年颁布了同样的软件产品质量评价标准 GB/T 16260—1996。ISO 9126 模型如图 10-3 所示。

ISO 9126 模型定义了 6 个影响软件质量的质量特性，每个质量特性又可通过若干子特性来测量，每个子特性在评价时要进行定义并实施若干度量。

ISO 9126 质量模型使得软件最大限度地满足用户明确的和潜在的需求，且从用户、开发人员、管理者等各类人员的角度全方位地考虑软件质量。



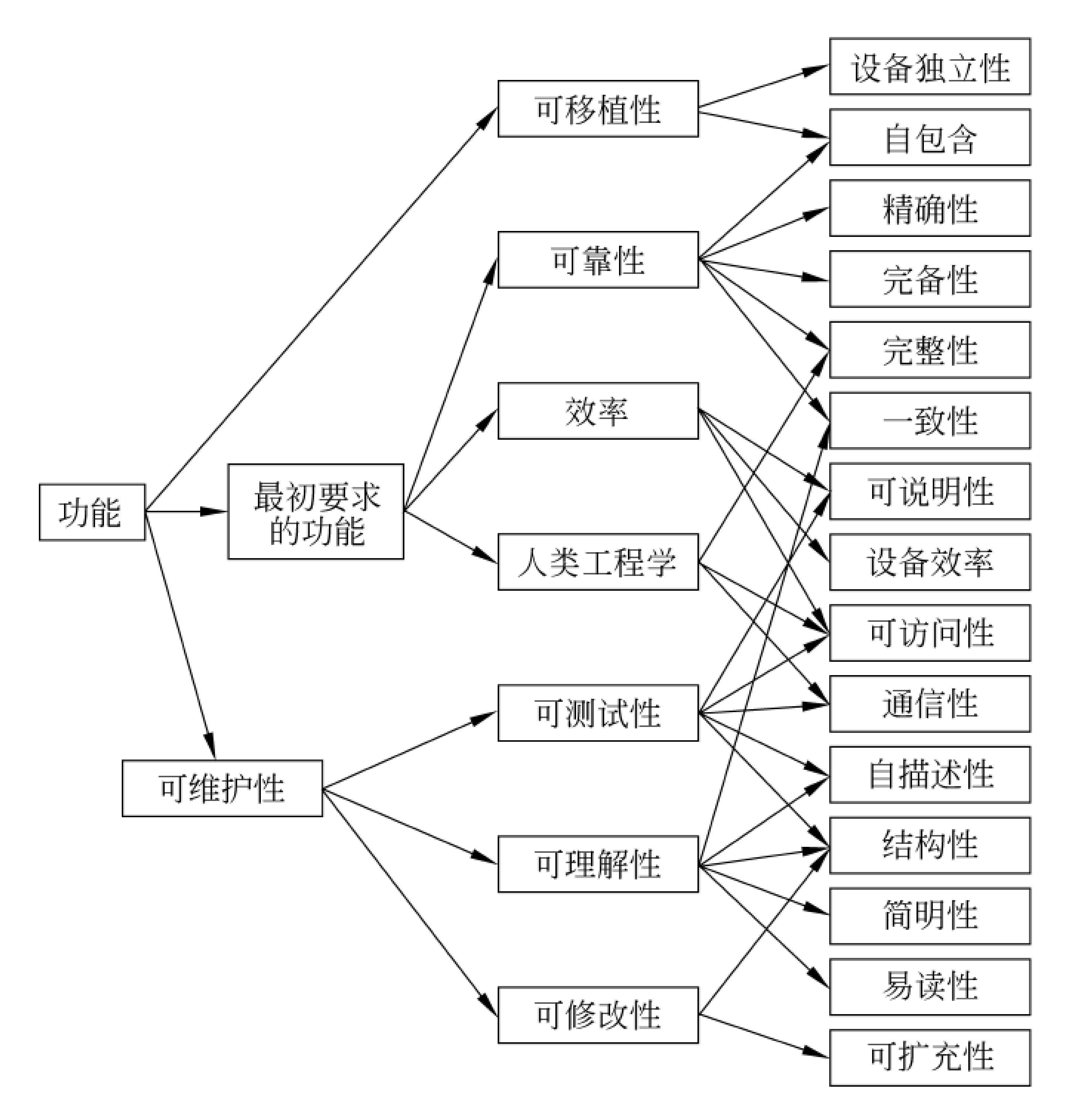


图 10-2 Boehm 质量模型

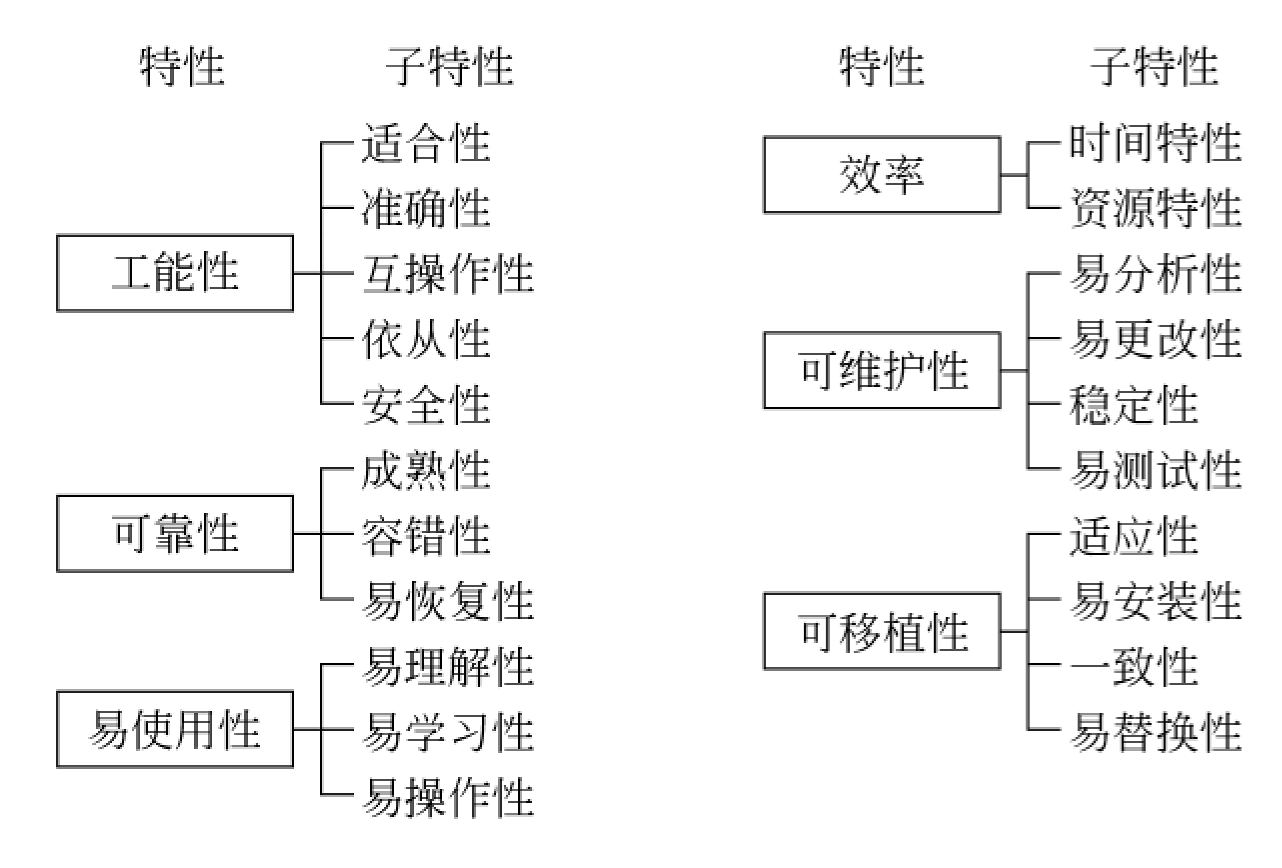


图 10-3 ISO 9126 质量模型

10.2 软件度量

10.2.1 软件度量概述

1. 软件度量的含义

在 10.1 节介绍软件质量模型时，曾多次讲到度量，究竟什么是度量呢？

度量的正式定义是：度量是指在现实的世界中把数字或符号指定给实体的某一属

性,以便以这种方式来根据已明确的规则描述它们。

从该定义可以看出,度量关注的是获取关于实体属性的信息。

那么什么是软件度量呢?软件度量(Software Measurement)是对软件开发项目、过程及其产品进行数据定义、收集以及分析的持续性量化过程,目的在于对项目质量、过程质量及产品质量进行理解、预测、评估、控制和改善。

软件度量或者说软件工程度量领域是一个在过去 30 多年研究非常活跃的软件工程领域。目前,软件研究人员及从业人员提出了上千种软件度量方法。

2. 软件度量的意义

在软件开发中,软件度量的根本目的是为了软件管理的需要,利用度量来改进软件过程,以提高软件开发效率和软件质量。人们是无法管理不能度量的事物的。通过软件度量,使人们能够可预测、可重复、准确地控制软件开发过程和软件产品。

度量使得对软件质量的评价从定性走向定量,避免了软件质量评价的模糊性。

3. 软件度量的现状

在软件界,目前软件度量的情况并不尽如人意,如下一些现象时常存在:

(1) 设计和开发软件产品的时候,并未制定出度量的目标。例如,软件开发机构保证将使软件的用户界面友好、可靠、易于维护,而并未使用度量的术语来详细说明它们的具体含义。Gilb 曾经说过:所谓模糊目标定理,就是没有明确目标的项目将不能明确地达到它的目标。

(2) 未能对构成软件项目实际费用的各个不同部分进行有效的度量。例如,通常开发团队并不知道,与测试阶段相比,设计阶段花费的时间是多少。

(3) 由于缺乏清晰的度量目标,使得开发人员不能使开发的产品质量的各方面特性都合格,因此不能使用术语向潜在的用户说明软件产品具有很高的质量。例如,在一段时间里使用故障的可能性、把产品安装到新环境中需花费的工作量等。

(4) 由于缺乏对软件的度量,看不到清晰的实效,因而人们对所使用的软件开发技术没有足够的信心,总是试图说服自己使用另一种新的技术进行软件开发。

事实上,目前人们在软件度量方面做的工作一般较少,而且所做的度量方面的工作也与一般科学意义上的度量存在很大距离。人们经常会看到诸如此类的话:“软件的费用有 80%花费在维护上”,或“软件每一千行程序中平均有 55 个 bugs”,但并不能知道这些结论是怎样产生的,试验是怎样设计并执行的,度量的是哪个实体,以及错误的框架是什么等。没有这些东西,就不能在自己的环境中客观地进行反复度量、重现度量的结果以获得与工业标准的真实比较。

4. 软件度量工具

随着软件定量方法的重要性不断增加,市场上出现了许多度量工具。然而,度量工具目前还是很混乱。因为没有统一的度量标准规范,每种工具发明商家都是按照他们自己的软件度量规范。Daich 等根据分类学把度量工具分成了以下几种:

- 通用度量工具。
- 小生境度量工具(Niche Metrics Tool)。

- 静态分析工具。
- 源代码静态分析工具。
- 规模度量工具。

10.2.2 软件度量的目标

度量活动必须有明确的目标或目的，而正是这决定着我们选择哪种属性和实体进行度量。这个目标与软件开发、使用时所涉及的人员的层次有关。

下面主要从管理者和软件工程师两种角度考虑，为了达到各种目标所要进行的度量工作。

1. 对管理者

(1) 需要度量软件开发过程中不同阶段的费用。

例如，度量开发整个软件系统的费用(包括从需求分析阶段到发布之后的维护阶段)。必须清楚这个费用以决定在保证一定利润的情况下的价格。

(2) 为了决定付给不同的开发小组的费用，需要度量不同小组职员的生产率。

(3) 为了对不同的项目进行比较，对将来的项目进行预测，建立基线及设定合理的改进目标等，需要度量开发产品的质量。

(4) 需要决定项目的度量目标。例如，应达到多大的测试覆盖率，系统最后的可靠性应有多大等。

(5) 为了找出是什么因素影响费用和生产率，需要反复测试某一特定过程和资源的属性。

(6) 需要度量和估计不同软件工程方法和工具的效用，以便决定是否有必要把它们引进公司。

2. 对软件工程师

(1) 需要制定过程度量以监视不断演进的系统。这包括设计过程中的改动，在不同的回顾或测试阶段发现的错误等。

(2) 需使用严格的度量术语来指定对软件质量和性能的要求，以便使这些要求是可测试的。例如，系统必须“可靠”，可用如下更具体的文字加以描述：“平均错误时间必须大于 15 个 CPU 时间片。”

(3) 为了合格需要度量产品和过程的属性。例如，一个产品是否合格要看该产品的一些可度量的特性，如“β 测试阶段少于 20 个错误”，“每个模块的代码行不超过 100 行”，以及开发过程的一些属性，如“单元测试必须覆盖 90%以上的用例”等。

(4) 需要度量当前已存在的产品和过程的属性，以便预测将来的产品。例如：

- 通过度量软件规格说明书的文档大小来预测目标软件的大小。
- 通过度量设计文档的结构特性来预测将来维护的盲点。
- 通过度量测试阶段的软件的可靠性来预测软件今后操作、运行的可靠性。

研究上面列出的度量目标和活动可以发现，软件度量的目标可大致概括为以下两类：

① 使用度量来进行估计。这使得人们可以同步地跟踪一个特定的软件项目。

② 应用度量来预测项目的一些重要的特性。注意，不能过分夸大这些预测的作用，

因为它们并不是完全正确的。

10.2.3　软件度量的内容

软件度量贯穿于软件开发的整个生命周期，是软件开发过程中进行理解、预测、评估、控制和改善的重要载体。软件度量包括项目度量、产品度量和过程度量三个方面，如表 10-1 所示。

表 10-1　软件度量的具体内容

度量维度	侧　重　点	具 体 内 容
项目度量	理解和控制当前项目的情况和状态	规模，成本，工作量，进度，生产力，风险，顾客满意度
产品度量	理解和控制当前产品的质量	产品的功能性，可靠性，易使用性，效率，可维护性，可移植性
过程度量	理解和控制当前的情况和状态，对过程进行改进和预测	能力成熟度，管理，生命周期，生产率，缺陷植入率

1. 项目度量

项目度量侧重于理解和控制整个软件项目的情况和状态，主要从软件项目的规模、成本、工作量、进度、生产力、风险、顾客满意度等方面进行度量。

(1) 规模度量

软件开发项目的规模度量（Size Measurement）是估算软件项目工作量、编制成本预算、策划合理的项目进度的基础。

有效的软件规模度量是项目成功的核心要素。这是因为基于有效的软件规模度量可以策划合理的软件项目计划，合理的项目计划则有助于有效地对项目进行管理。

软件规模度量的要点如下：

① 由开发现场的项目成员进行估算。

② 灵活运用实际开发作业数据。

③ 杜绝盲目迎合顾客需求的“交期逆推法”。

软件规模度量有助于软件开发团队准确地把握开发时间、费用分布及缺陷密度等。软件规模的估算方法有很多种，例如功能点分析（Function Points Analysis，FPA），代码行（Lines of Code，LOC），德尔菲法（Delphi Technique），COCOMO 模型，特征点（Feature Point），对象点（Object Point），3-D 功能点（3-D Function Points），Bang 度量（DeMarco's Bang Metric），模糊逻辑（Fuzzy Logic），标准构件法（Standard Component）等，这些方法不断细化为更多具体的方法。

(2) 成本度量

软件开发的成本度量主要指软件开发项目所需财务性成本的估算，主要方法如下：

① 类比估算法。类比估算法是通过比较已完成的类似项目系统来估算成本。它适合于评估一些与历史项目在应用领域、环境和复杂度方面相似的项目。其约束条件是必须存在类似的、具有可比性的软件开发系统，估算结果的精确度依赖于历史项目数据的完整性、准确度，以及现行项目与历史项目的近似程度。

② 细分估算法。细分估算法是将整个项目系统分解成若干个小系统，逐个估算成本，然后合计起来作为整个项目的估算成本。细分估算法通过逐渐细化的方式对每个小系统进行详细的估算，可能获得贴近实际的估算成本。其难点在于难以把握各小系统整合为大系统的整合成本。

③ 周期估算法。周期估算法是按软件开发周期进行划分，估算各个阶段的成本，然后进行汇总合计。周期估算法基于软件工程理论对软件开发的各个阶段进行估算，适合于瀑布型软件开发方法，但是需要估算者对软件工程各个阶段的作业量和相互间的比例具有相当的了解。

(3) 顾客满意度度量

顾客满意是软件开发项目的主要目的之一，而顾客满意目标要得以实现，需要建立顾客满意度度量体系和指标，对顾客满意度进行度量。顾客满意度指标(Customer Satisfaction Index，CSI)以顾客满意度研究为基础，对顾客满意度加以界定和描述。

项目顾客满意度量的要点是，确定各类信息、数据、资料来源的准确性、客观性、合理性和有效性，并以此建立产品、服务质量的衡量指标和标准。

Stephen H. Kan 在《软件质量工程的度量与模型》(Metrics and Models in Software Quality Engineering)一书中描述的企业的顾客满意度要素及其内容如表 10-2 所示。

表 10-2 顾客满意度要素及其内容

顾客满意度要素	顾客满意度要素的内容
技术解决方案	质量，可靠性，有效性，易用性，价格，安装，新技术
支持与维护	灵活性，易达性，产品知识
市场营销	解决方案，接触点，信息
管理	购买流程，请求手续，保证期限，注意事项
交付	准时，准确，交付后过程
企业形象	技术领导，财务稳定性，执行印象

作为企业的顾客满意度的基本构成单位，项目的顾客满意度会受到项目要素的影响，主要包括开发的软件产品、开发文档、项目进度及交付期、技术水平、沟通能力、运用维护等。具体而言，可细分为如表 10-3 所示的度量要素，并可根据这些要素进行度量。

表 10-3 顾客满意度项目的度量要素

顾客满意度项目	顾客满意度度量要素
软件产品	功能性，可靠性，易用性，效率性，可维护性，可移植性
开发文档	文档的构成，质量，外观，图表及索引，用语
项目进度及交付期	交付期的根据，进度迟延情况下的应对，进展报告
技术水平	项目组的技术水平，项目组的提案能力，项目组的问题解决能力
沟通能力	事件记录，式样确认，Q&A
运用维护	支持，问题发生时的应对速度，问题解决能力

2. 产品度量

软件产品的度量主要针对于作为软件开发成果的软件产品的质量而言，独立于其过程。软件的质量由一系列质量要素组成，每一个质量要素由一些衡量标准组成，每个衡量标准又由一些量度标准加以定量刻画。质量度量贯穿于软件工程的全过程及软件交付之后，在软件交付之前的度量主要包括程序复杂性、模块的有效性和总的程序规模，在软件交付之后的度量则主要包括残存的缺陷数和系统的可维护性。

前面介绍过的软件质量模型实际上就是软件产品度量的依据，如 McCall 质量模型、Boehm 质量模型及 ISO 9126 质量模型。这些模型指出了对于模型中的每一个质量要素具体的度量标准，可用来定量地评价软件产品的质量，在此不详细介绍。

3. 过程度量

过程度量是对软件开发流程各个方面进行的度量，目的在于预测过程的未来性能，减少过程结果的偏差，对软件过程的行为进行目标管理，为过程控制、过程评价及持续改善提供定量基础。

软件过程质量的好坏将直接影响软件产品质量的好坏，度量并评估过程、提高过程成熟度可以改进产品质量。相反，度量并评估软件产品质量会为提高软件过程质量提供必要的反馈和依据。

软件过程度量主要包括以下三方面的内容。

(1) 成熟度度量(Maturity Metrics)

主要包括组织度量、资源度量、培训度量、文档标准化度量、数据管理与分析度量、过程质量度量等。

(2) 管理度量(Management Metrics)

主要包括项目管理度量(如里程碑管理度量、风险度量、作业流程度量、控制度量、管理数据库度量等)、质量管理度量(如质量审查度量、质量测试度量、质量保证度量等)，配置管理度量(如式样变更控制度量、版本管理控制度量等)。

(3) 生命周期度量(Life Cycle Metrics)

主要包括问题定义度量、需求分析度量、设计度量、制造度量、维护度量等。

10.3 软件能力成熟度模型

10.3.1 软件能力成熟度模型概述

1. 软件能力成熟度模型的起源

早在 20 世纪 70 年代中期，美国国防部就组织力量研究软件项目失败的原因，发现在失败的软件项目中，70%是由于管理不善所造成的，因而得出这样的结论：管理影响软件项目的全局，而技术只影响项目的局部。随后，在美国掀起了研究软件管理技术的热潮。

为了系统地解决软件项目管理问题，美国国防部于 1984 年在卡内基·梅隆大学建立了软件工程研究所。卡内基·梅隆大学软件工程研究所 CMU/SEI(Carnegie Mellon University/ Software Engineering Institute)于 1987 年研究发布了软件过程成熟度框架，

并提供了软件过程评估和软件能力评价两种评估方法和软件成熟度提问单。4 年之后，SEI 将软件过程成熟度框架进化为软件能力成熟度模型(Capability Maturity Model For Software，SW-CMM)，并发布了最早的 SW-CMM 1.0 版。经过两年的试用，1993 年 SEI 正式发布了 SW-CMM 1.1 版，这是目前使用最为广泛的版本。SW-CMM 通常简称为 CMM。

2. 软件能力成熟度模型中的重要概念

在对 CMM 深入介绍前，先介绍如下概念。

(1) 过程(Process)

CMM 中引用了 IEEE 对过程的定义，即过程是“为达到目的而执行的所有步骤的系列”。

(2) 软件过程(Software Process)

指人们用于开发和维护软件及其相关产品的一系列活动、方法、实践和革新。软件和其相关产品是指项目计划、需求文档、设计文档、代码、测试用例、用户手册等。

(3) 软件过程能力(Software Process Capability)

在遵循一个软件过程后能得到的预期结果的范围。它可用来预测一个机构在承接一个软件项目后，所能得到的最可能的结果。

(4) 软件过程性能(Software Process Performance)

在遵循一个软件过程后所得到的实际结果。

(5) 软件过程成熟度(Software Process Maturity)

一个具体的软件过程被明确定义、管理、评价、控制和产生实效的程度。所谓成熟度，包含着能力的一种增长潜力，同时也表明了软件机构实施软件过程的实际水平。

3. 软件能力成熟度模型的作用

软件能力成熟度模型 CMM 融合了全面质量管理的思想，是一个率先在软件行业从软件过程能力的角度提出的软件评估标准，通过它能够评价一个软件机构的开发过程的成熟度。

CMM 的目的是帮助软件企业对软件过程进行管理和改进，增强开发与改进能力，从而高效率地开发出高质量的软件。CMM 已成为业界事实上的软件过程的工业标准。

企业实施 CMM 模型并评估可为企业带来许多好处。例如：

- 指导软件机构提高软件开发管理能力；
- 降低软件承包商和采购者的风险；
- 评估软件承包商的软件开发管理能力；
- 帮助软件企业识别开发和维护软件的有效过程和关键实践；
- 帮助软件企业识别为达到 CMM 更高成熟等级所必须的关键实践；
- 增加软件企业的国际竞争能力。

4. 软件能力成熟度模型的 5 个等级

CMM 将软件过程的成熟度分为 5 个等级，如图 10-4 所示。

表 10-4 列出了 CMM 中各个等级的软件机构的特征。

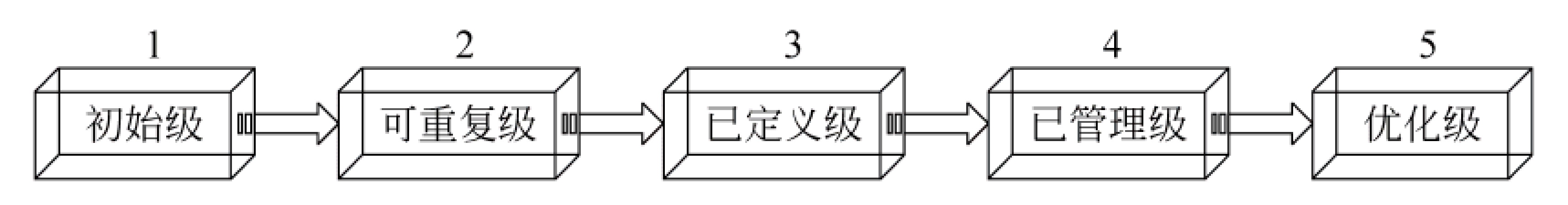


图 10-4　软件过程成熟度的 5 个等级

表 10-4　CMM 中各等级软件机构的特征

级　别	特　征
初始级(第 1 级)	① 对过程几乎没有定义,导致软件过程混乱无序、无法预测和控制 ② 项目的成功依靠个人的能力和行为 ③ 管理方式属于反应式
可重复级(第 2 级)	① 管理制度化,工作有章可循 ② 开发工作初步实现标准化 ③ 变更基线化 ④ 过程可跟踪 ⑤ 新项目计划和管理基于过去的实践经验,具有重复以前成功项目的环境和条件
已定义级(第 3 级)	① 开发过程标准化、文档化 ② 完善的培训和专家评审制度 ③ 技术和管理活动稳定实施 ④ 项目质量、进度和费用可控制 ⑤ 项目过程、岗位和职责均有共同的理解
已管理级(第 4 级)	① 产品和过程有定量的质量目标 ② 过程的生产率和质量可度量 ③ 有过程数据库 ④ 实现项目产品和过程的控制 ⑤ 过程和产品质量趋势可预测,若预测有偏差,可以及时纠正
优化级(第 5 级)	① 不断改进过程,采用新技术、新方法 ② 有防止缺陷、识别薄弱环节并加以改进的手段 ③ 可通过反馈获得过程有效性的统计数据,并进行分析,以改善过程

在 CMM 中,等级越高,软件开发生产精度越高,每单位工程的生产周期也越短。

在 CMM 的 5 个等级中,除了初始级,其他 4 个等级均有若干个指导软件机构改进软件过程的要点,称为关键过程域(Key Process Area,KPA)。一个软件机构,如果希望达到某一个成熟度等级,就必须完全满足关键过程域所规定的不同要求,即满足每个关键过程域的目标。所谓关键过程域,是指一系列相互关联的操作活动,这些活动反映了一个软件机构改进过程中必须集中精力改进的方面。只有一个软件机构的所有项目都达到某个关键过程域的目标,该软件机构以该关键过程域为特征的过程能力才是规范化的。

在 CMM 中一共有 18 个关键过程域,分布在 2～5 级中。表 10-5 列出了 CMM 中各等级对应的关键过程域和主要活动。

表 10-5 CMM 中各等级对应的关键过程域和主要活动

级　　别	关键过程域	主 要 活 动
初始级	无	
可重复级	① 需求管理(Requirement Management,RM):在客户和软件项目之间达成对客户需求的一致理解 ② 软件项目策划(Software Project Planning,SPP):为软件工程和项目管理建立一个合理的计划 ③ 软件项目跟踪和监控(Software Project Tracking and Oversight,SPTO):使管理者对实际的软件项目进展过程有足够的了解,以在项目效能偏离计划较多时能够采取有效措施 ④ 软件子合同管理(Software Subcontract Management,SSM):选择合格的分包商,并有效管理之 ⑤ 软件质量保证(Software Quality Assurance,SQA):对软件项目过程及其间生产的各个产品进行监管以保证最终软件质量 ⑥ 软件配置管理(Software Configuration Management,SCM):在整个软件生命周期里建立并维护软件项目的工作产品的完整性。	主要涉及建立软件项目管理控制方面的活动
已定义级	① 组织过程聚焦(Organization Process Focus,OPF):确立机构对于改进机构的软件过程能力的软件过程活动的责任 ② 组织过程定义(Organization Process Definition,OPD):开发和维护一组有用的软件过程资产,提供一个用于定义定量过程管理的有意义的数据的基础 ③ 培训大纲(Training Program,TP):开发个体的技能和知识,以使他们能够更好地扮演自己的角色 ④ 集成软件管理(Integrated Software Management,ISM):基于业务环境和项目的技术需要,从机构的标准软件过程和相关的过程资产经过剪裁,将软件工程和管理活动集成为一个有机的定义的软件过程 ⑤ 组间协调(Intergroup Coordination,IC):确立软件工程组主动介入其他工程组以便项目能更好地满足客户要求的手段 ⑥ 软件产品工程(Software Product Engineering,SPE):一致地完成定义良好的工程过程,它描述了项目的技术活动,如需求分析、设计、编码和测试 ⑦ 同行评审(Peer Reviews,PR):通过同行专家,尽早且有效地排除软件产品中的缺陷	主要涉及项目和组织的策略,使软件组织建立起对项目中的有效计划和管理过程的内部细节
已管理级	① 定量过程管理(Quantitative Process Management,QPM):定量地控制软件项目的过程效能 ② 软件质量管理(Software Quality Management,SQM):定量了解软件产品的质量,并达到既定的质量目标	主要的任务是为软件过程和软件产品建立一种可以理解的定量的方式
优化级	① 过程变更管理(Process Change Management,PCM):连续改进机构所采用的软件过程,以改进软件质量,提高生产率和减少产品开发时间 ② 缺陷预防(Defect Prevention,DP):识别出错原因,防止错误再现(通过改变定义的软件过程) ③ 技术变更管理(Technology Change Management,TCM):识别有益的新技术(工具、方法和过程),并按有序的方式将其转移至机构之中	主要涉及软件组织和项目中如何实现持续不断的过程改进

值得一提的是,任何一个成熟度级别的关键过程域集都是本级描述的关键过程域集和所有下级的关键过程域集的并集。例如第 3 级的关键过程域就应有 13 个不同的域,其中 7 个是第 3 级自己包含的,6 个属于第 2 级,而第 4 级应有 15 个域。

5. 关键实践和共同特征

为了达到某个成熟度级别的关键过程域的目标,必须实施若干关键实践。所谓关键实践(Key Practice),是指对关键过程域起重要作用的方针、规程、措施、活动,以及相关基础设施的建立。关键实践涉及 5 个方面,分别为执行约定、执行能力、执行的活动、测量和分析、验证实施,被称作 5 个共同特征(Common Features)。每个关键过程域所包含的关键实践均按这 5 个共同特征进行组织。

共同特征是表明一个关键过程域的实施和规范化是否有效、可重复且持久的一些属性。这 5 个共同特征的含义如下。

(1) 执行约定(Commitment to Perform)

描述一个机构在保证将过程建立起来并持续起作用方面所必须采取的行动。执行约定一般包括制定机构的方针和规定高级管理人员的支持。

(2) 执行能力(Ability to Perform)

描述为了实施软件过程,项目或机构中必须存在的先决条件。一般包括资源、组织机构和培训。

(3) 执行的活动(Activities Performed)

描述为了实现一个关键过程域所需要的角色和规程。执行的活动一般包括制订计划与规程、执行计划、跟踪执行情况,必要时采取纠正措施。

(4) 测量和分析(Measurement and Analysis)

描述对过程进行测量和对测量结果进行分析的需要。测量和分析一般包括为了确定所执行活动的状态及有效性所能采用的测量和分析。

(5) 验证实施(Verifying Implementation)

描述遵照已建立的过程进行活动的措施。验证一般包括管理人员和软件质量保证部门所做的评审和审核。

6. CMM 的逻辑结构

概括的 CMM 逻辑结构如图 10-5 所示。

图 10-5　CMM 的逻辑结构

10.3.2　软件能力成熟度模型的建立和评估

1. **CMM 的实施步骤**

企业实施 CMM,可从如下步骤进行:

① 提高思想认识,了解其必要性和迫切性。

② 确定合理的目标。

③ 进行 CMM 培训和咨询工作。
④ 成立工作组。
⑤ 制定和完善软件过程。
⑥ 内部评审。
⑦ 初期评估。
⑧ 正式评估。
⑨ 根据评估的结果改进软件过程。

2. CMM 的评估方法

CMM 的评估方法为 CBA-IPI（CMM-Based Appraisal for Internal Process Improvement，基于 CMM 的内部过程改进评估）。CMM 评估包括 5 个等级，共计 18 个关键过程域，52 个目标，300 多个关键实践。每一个 CMM 等级评估周期约需 12～30 个月。每一级别的评估由 CMU/SEI 授权的主任评估师领导一个评审小组进行。评估过程包括员工培训、问卷填写和统计、文档审查、数据分析、与企业的高层领导讨论和撰写评估报告等。

3. 实施 CMM 应把握的原则

应该注意的是，任何软件开发单位在提升自己的 CMM 等级时，只能由所处的层次向紧邻的上一层次进化，不可越级。而且在由某一成熟层次向上一更成熟层次进化时，在原有层次中的那些已经具备的能力还必须得到保持与发扬。

此外，通过 CMM 某个等级的评估并不是软件企业最终的目的，而最终目的应是通过 CMM 评估来改进企业的软件过程，提升软件开发的能力。

10.3.3 个体软件过程 PSP 和群组软件过程 TSP

由于 CMM 并未提供有关实现 CMM 关键过程域所需的具体知识和技能，因此 CMU/SEI 于 1989 年开始研究，并于 1994 年提出个体软件过程（Personal Software Process，PSP）。1994 年开始研究并于 1998 年由 CMU/SEI 召开的过程工程年会上第一次介绍了群组软件过程（Team Software Process，TSP）草案，于 1999 年出版了有关 TSP 的一本书，使软件过程框架形成一个包含 CMM，PSP 和 TSP 三者的严密的整体，形成 CMM/PSP/TSP 体系。

个体软件过程 PSP 是一种可用于控制、管理和改进个人工作方式的自我改善过程，是一个包括软件开发表格、指南和规程的结构化框架。PSP 为基于个体和小型群组软件过程的优化提供了具体而有效的途径，例如如何制订计划，如何控制质量，如何与其他人相互协作，如何建立度量个体软件过程改善的基准等。

在软件设计阶段，PSP 的着眼点在于软件缺陷的预防，具体办法是强化设计结束准则，而不是设计方法的选择。PSP 保障软件产品质量的一个重要途径是提高设计质量。

经过 PSP 学习和实践的正规训练，软件工程师们能够在他们参与的项目工作中充分运用 PSP，从而有助于 CMM 目标的实现。

群组软件过程（Team Software Process，TSP）结合了 CMM 的管理方法和 PSP 的工程技能，指导项目组中的成员如何有效地规划和管理所面临的项目开发任务，并且告诉管

理人员如何指导软件开发队伍，始终以最佳状态来完成工作。TSP 实施集体管理与自己管理相结合的原则，最终目的在于指导开发人员如何在最少的时间内，以预定的费用生产出高质量的软件产品，所采用的方法是对群组开发过程的定义、度量和改进。

目前认为 TSP 比较适合规模为 3～20 人的开发小组。

CMM，PSP 和 TSP 为软件产业提供了一个集成化的、三维的软件过程改革框架。在软件项目开发过程中，应将实施 CMM 与实施 PSP，TSP 有机地结合起来，才能达到软件过程持续改善的效果。

10.3.4　能力成熟度模型集成 CMMI

1. CMMI **概述**

自从 1991 年 SEI 正式发布软件 SW-CMM 以来，相继又开发出了系统工程、软件采购、人力资源管理及集成产品开发方面的多个能力成熟度模型。虽然这些模型在许多组织都得到了良好的应用，但对于一些大型软件企业来说，可能会出现需要同时采用多种模型来改进自己多方面过程能力的情况。这时他们就会发现存在一些问题，其中主要问题体现在下面几方面：

- 不能集中其不同过程改进的能力以取得更大成绩。
- 要进行一些重复的培训、评估和改进活动，因而增加了成本。
- 遇到不同模型中有一些对相同事物说法不一致，或活动不协调，甚至相抵触。

所以，当 CMU/SEI 开始开发新一代成熟度模型的时候，试图整合不同模型中的最佳实践，建立统一模型，覆盖不同领域，供企业进行整个组织的全面过程改进。CMU/SEI 于 2001 年 12 月正式发布了能力成熟度集成模型（Capability Maturity Model Integration，CMMI）1.1 版，这标志着 CMMI 的正式使用。CMU/SEI 也正式宣布，将不再维护 SW-CMM 的 CBA-IPI 评估方法：在 CMMI 1.1 发布后的两年内，还提供有关 SW-CMM 和 CBA-IPI 主任评估员的培训，并接收评估数据，但这一切已于 2003 年 12 月底正式停止。

值得注意的是，SEI 并没有废除 CMM 模型，而是以 CMMI 的 SCAMPI（Standard CMMI Appraisal Method for Process Improvement）评估方法取代 CMM 的 CBA-IPI 评估方法。当然，随着软件行业的发展，CMMI 模型将最终取代 CMM 模型。这是因为，相比于 CMM，CMMI 以更加系统和一致的框架来指导组织改进软件过程，提高产品和服务的开发、获取和维护能力。

CMMI 起源于软件能力成熟度模型（SW-CMM） v2.0 draft C、电子行业协会临时标准（EIA/IS-731）和集成产品开发能力成熟度模型（IPD-CMM）v0.98 三个模型（源模型）。模型中同时集成了供应管理的内容。另外，在 CMMI 中除了沿用 CMM 的分阶段表示形式外，还增加了与 ISO/IEC 15504 TR（技术报告）类似的连续表示形式。连续式表示形式（即 CMMI 连续式模型）将 CMMI 中的过程区域分为四大类：项目管理、工程活动、支持活动或者过程管理，对于每个大类中的过程区域，又进一步分为基本的和高级的。这样，按照连续式模型实施 CMMI 的时候，一个组织可以把某个大类（如项目管理）的实践一直做到最好，而其他大类的实践则完全不考虑。也就是说，如果企业只希望提高自己在

项目管理、工程活动、支持活动或者过程管理四个大类中的某一类或几类的能力，则应使用 CMMI 的连续表示形式。若企业仍愿沿用 CMM 中成熟度级别的思路，持续地改进过程，则可使用 CMMI 的分阶段表示形式（即 CMMI 分级式模型）。

CMMI 模型的组成和适用范围如表 10-6 所示。

表 10-6　CMMI 模型的组成和适用范围

CMMI 模型组件	适 用 范 围
SE/SW	系统工程，软件工程
SE/SW /IPPD	系统工程，软件工程，集成产品和过程开发
SE/SW /IPPD/SS	系统工程，软件工程，集成产品和过程开发、供应采购
培训课程	评估师、过程改进人员等培训
SCMPI	评估方法

在 CMMI 模型组件中，SE/SW 是核心，SE/SW/IPPD，SE/SW/IPPD/SS 是在此基础上扩展而来的。

2. CMMI 的主要内容

CMMI 分级式模型中的等级设置与 SW-CMM 模型基本相同，只是某些等级的名称有变化。

CMMI 连续式模型中的等级设置则与 ISO/IEC 15504 TR 基本一致，CMMI 连续式模型的第 3 级、第 4 级名称虽然与 ISO/IEC 15504 TR 有区别，但其含义是基本一样的。

表 10-7 列出了 CMMI，CMM，ISO/IEC 15504 TR 模型中等级的对应关系。

表 10-7　各模型的等级对应关系

等级	CMM	CMMI（分级式）	CMMI（连续式）	ISO/IEC 15504 TR
5	优化极	优化级	优化级	优化级
4	已管理级	定量管理级	定量管理级	可预测级
3	已定义级	已定义级	已定义级	已建立级
2	可重复级	已管理级	已管理级	已管理级
1	初始级	初始级	已执行级	已执行级
0			未完成级	未完成级

下面对 CMMI 分级式模型和 CMM 各等级中的关键过程域进行比较。

CMMI 共有分别属于 4 个等级的 25 个关键过程域。CMMI 对 CMM 的关键过程域主要进行了如下变更。

CMM 第 2 级共有 6 个关键过程域，CMMI 则增加了 1 个：度量和分析。原先 6 个关键过程域的名称和内容在 CMMI 中做了部分改进，但是主体内容没有大幅调整。

CMM 第 4 级共有 2 个关键过程域，在 CMMI 中仍是 2 个，只是名称和内容有所改进。

CMM 第 5 级共有 3 个关键过程域，在 CMMI 中进行了合并，改为 2 个，但主要内容未变。

变化最显著的是在第 3 级上，CMM 中 7 个关键过程域变成了 CMMI 中的 14 个，其中原来对工程活动进行要求的关键过程域——软件产品工程进行了详细的拆分，并结合常见的软件生命周期模型进行了映射。CMMI 中新增的关键过程域中还涉及过去未曾提到的内容，比如决策分析和决定、集成组队等。

两个模型的关键过程域比较见表 10-8。

表 10-8　CMMI 和 CMM 的关键过程域比较

<table>
<tr><th rowspan="2">等级</th><th colspan="2">CMM</th><th colspan="2">CMMI</th><th rowspan="2">类　别</th></tr>
<tr><th>关键过程域</th><th>缩写</th><th>关键过程域</th><th>缩写</th></tr>
<tr><td rowspan="3">5</td><td>技术变更管理</td><td>TCM</td><td rowspan="2">组织革新与部署</td><td rowspan="2">OID</td><td rowspan="2">过程管理</td></tr>
<tr><td>过程变更管理</td><td>PCM</td></tr>
<tr><td>缺陷预防</td><td>DP</td><td>原因分析与决策</td><td>CAR</td><td>支持</td></tr>
<tr><td rowspan="2">4</td><td>软件质量管理</td><td>SQM</td><td>组织过程性能</td><td>OPP</td><td>过程管理</td></tr>
<tr><td>定量过程管理</td><td>QPM</td><td>定量项目管理</td><td>QPM</td><td>项目管理</td></tr>
<tr><td rowspan="16">3</td><td rowspan="4">软件产品工程</td><td rowspan="4">SPE</td><td>需求制定</td><td>RD</td><td>工程</td></tr>
<tr><td>技术方案</td><td>TS</td><td>工程</td></tr>
<tr><td>产品集成</td><td>PI</td><td>工程</td></tr>
<tr><td>验证</td><td>VER</td><td>工程</td></tr>
<tr><td>同行评审</td><td>PR</td><td>确认</td><td>VAL</td><td>工程</td></tr>
<tr><td>组织过程聚焦</td><td>OPF</td><td>组织过程聚焦</td><td>OPF</td><td>过程管理</td></tr>
<tr><td>组织过程定义</td><td>OPD</td><td>组织过程定义</td><td>OPD</td><td>过程管理</td></tr>
<tr><td>培训大纲</td><td>TP</td><td>组织培训</td><td>OT</td><td>过程管理</td></tr>
<tr><td>集成软件管理</td><td>ISM</td><td rowspan="2">集成项目管理</td><td rowspan="2">IPM</td><td rowspan="2">项目管理</td></tr>
<tr><td>组间协调</td><td>IC</td></tr>
<tr><td></td><td></td><td>风险管理</td><td>RSKM</td><td>项目管理</td></tr>
<tr><td></td><td></td><td>决策分析与决定</td><td>DAR</td><td>支持</td></tr>
<tr><td></td><td></td><td>集成供应商管理</td><td>ISM</td><td>项目管理</td></tr>
<tr><td></td><td></td><td>组织集成环境</td><td>OEI</td><td>支持</td></tr>
<tr><td></td><td></td><td>集成组队</td><td>IT</td><td>项目管理</td></tr>
<tr><td rowspan="3">2</td><td>需求管理</td><td>RM</td><td>需求管理</td><td>RM</td><td>工程</td></tr>
<tr><td>软件项目策划</td><td>SPP</td><td>项目策划</td><td>PP</td><td>项目管理</td></tr>
<tr><td>软件项目跟踪与监控</td><td>SPTO</td><td>项目监督与控制</td><td>PMC</td><td>项目管理</td></tr>
</table>

续表

等级	CMM		CMMI		类　别
	关键过程域	缩写	关键过程域	缩写	
2	软件子合同管理	SSM	供应协议管理	SAM	项目管理
	软件质量保证	SQA	过程与产品质量保证	PPQA	支持
	软件配置管理	SCM	配置管理	CM	支持
			度量与分析	MA	支持
1					

3. CMMI 的评估方法

随着 CMM 过渡到 CMMI，其 CAF（CMM Assessment Framework）评估框架变成评估需求（Appraisal Requirements for CMMI，ARC），CMM 的 IPI-CBA 评估方法则被 CMMI 的 SCAMPI 评估方法所取代。

开展 CMMI 评估必须是 SEI 授权的合作机构，评估活动由 SEI 授权的主任评估师进行。CMMI 的评估流程与 CMM 基本相同。

另外，与 CMM 中的等级评估相同的是，CMMI（分级式模型）中的等级评估会根据被评估的成熟度级别，检查所有不高于该级别的关键过程域。

4. 如何在 CMM 和 CMMI 之间进行选择

究竟是选择 CMM 还是 CMMI，主要应基于以下方面进行考虑。

（1）实施企业的业务特点

如果企业的规模不是很大，经费有限，业务又以软件开发为主，那么还是 CMM 比较适用。如果企业的规模比较大（开发人员 100 人以上），并且业务不仅仅集中在软件开发，还包括硬件开发哪怕是硬件代理（采购）都可以考虑实施 CMMI。

（2）实施企业对过程改进的熟悉程度

如果企业已经实施过 ISO 9000，并且取得了较好的效果，那么可以考虑实施 CMMI。如果企业虽然没有实施过 CMM，但是对于过程改进一直比较关注，接受过一些相关培训，甚至能够自发地进行一些过程改进，那么也可以考虑实施 CMMI。如果过去没有接触过类似的工作，那么最好先从软件 CMM 2 级开始，建立持续过程改进的思路，以后再向 CMMI 平滑地转换。

（3）软件过程改进的目标

如果企业只希望单方面地提高自己在项目管理、工程活动、支持活动或者过程管理 4 个方面中的某个或某几个方面的能力，就应使用 CMMI（CMMI 连续式模型）；如果企业过程改进仍沿用 CMM 中成熟度级别的思路，希望全面地提升过程能力，则不一定非要使用 CMMI。

10.3.5 SPCA 评估体系

为了提高国内软件企业的竞争力，我国信息产业部会同国家认证认可监督委员会，在

研究国际软件评估体制尤其是 CMMI 的基础上，考虑国内软件产业实际情况，于 2001 年建立了 SPCA（软件过程及能力成熟度评估，又称“双模认证”）评估体系。

SPCA 是软件过程能力评估和软件能力成熟度评估的统称。SPCA 所依据的评估标准是我国制定的 SJ/T 11234《软件过程能力评估模型》和 SJ/T 11235《软件能力成熟度模型》。

10.4　其他软件管理体系

除了 CMM/CMMI 外，应用较广泛的软件管理体系还有 ISO 9000 质量标准体系和 ISO/IEC 15504 标准。

ISO 9000 是国际标准化组织（International Organization for Standardization，ISO）颁布的全世界范围内通用的质量管理和质量保证方面的一系列标准。

ISO/IEC 15504（信息技术-过程评估）标准则是国际标准化组织和国际电工委员会（IEC）在 ISO/IEC 15504 TR（SPICE）的基础上制定的正式国际标准，用于软件过程的评估。

关于 ISO 9000 质量标准体系和 ISO/IEC 15504 标准，目前的介绍很多，读者可参看相关资料尤其是网站资料，以比较与 CMM/CMMI 的异同。

10.5　软件质量保证

10.5.1　软件质量保证概述

软件质量保证 SQA（Software Quality Assurance）是 CMM 第 2 级中的一个关键过程域，它是贯穿整个软件过程的第三方独立审查活动，出现在大多数关键过程域的检查与验证的公共特性中，在整个软件开发过程中充当重要角色。

实践证明，软件质量保证活动在提高软件质量方面卓有成效。IBM 360/370 系统软件的开发经验证明了这一点。IBM 的有关报告指出，在 8 年的时间里，软件质量提高了 3～5 倍，而 SQA 是其质量体系中的一个重要组成部分。

1. 软件质量保证的目的

SQA 的目的是向管理者提供对软件过程进行全面监控的手段，包括评审和审计软件产品和活动，验证它们是否符合相应的规程和标准，同时给项目管理者提供这些评审和审计结果，以反映产品和过程质量，提高项目透明度。

值得注意的是，SQA 组织并不负责生产高质量的软件产品和制订质量计划，这些都属于软件开发人员的工作。

2. SQA 组织的主要工作

SQA 组织的主要工作包括两方面：监控软件的开发过程，保证软件开发过程符合相应的标准与规程；保证软件产品、软件过程中存在的不符合问题得到处理，必要时将问题反映给高级管理者。

除此以外，SQA 组织还可作为软件工程过程小组（Software Engineering Process Group，SEPG）在项目组中的延伸，能够收集项目中好的实施方法和发现实施不利的原因，为修改企业内部软件开发整体规范提供依据，为其他项目组的开发过程实施提供先进方法和样例。

注意，软件工程过程小组 SEPG 的职责是提供软件过程的指导，帮助项目组制定项目过程，实施过程改进。如果项目组和 SQA 对过程的理解不一致，SEPG 将作为最终仲裁者。可用立法者和执法者来形容 SEPG 与 SQA 之间的关系。

SQA 的主要作用是给管理者提供实现软件过程的保证，因此 SQA 组织需要保证：

- 选定的开发方法被采用；
- 选定的标准和规程得到采用和遵循；
- 进行独立的审查；
- 偏离标准和规程的问题得到及时地反映和处理；
- 项目定义的每个软件任务得到实际执行。

10.5.2 软件质量保证的工作内容

1. 计划

应该针对项目制定 SQA 计划，注意不是项目计划。制定 SQA 计划应当注意如下内容：

（1）确定软件质量保证活动的目标和审计内容。

（2）明确审计方式。

（3）确定审计结果报告的规范。

对于一般性项目，可采用通用的软件质量保证计划，而对于那些有着特殊质量要求的项目，则必须根据项目自身的特点制定专门的计划。

2. 审计/证实

依据 SQA 计划进行 SQA 审计工作，按照规范发布审计结果报告。

需要注意的是，审计一定要有项目组人员陪同，不能搞突然袭击。

审计的内容主要为项目组是否按照过程要求执行了相应活动，是否按照过程要求产生了相应产品。

3. 处理不符合问题

对审计过程中发现的不符合问题，SQA 人员应报告其不符合的地方，以及它对产品的影响，同时向项目组提出改进建议，并持续跟踪直到问题解决。若有项目组暂时无法解决的不符合问题，SQA 应向高级管理者直接反映。

10.5.3 软件质量保证的实施

软件质量保证的实施过程中需要注意如下方面的问题：

（1）企业的高级管理者必须重视软件质量保证活动。在一些组织的软件生产过程中，高级管理者不重视软件质量保证活动，对 SQA 人员发现的问题不及时处理。如此一来，软件质量保证就流于形式，很难发挥它应有的作用。

(2) 要考虑 SQA 人员的素质。SQA 人员的素质主要体现在熟悉软件开发过程及企业内部已有的开发过程规范;掌握专业的技术,例如质量控制知识、统计学知识等;有很强的沟通能力。为提高 SQA 人员的素质,可对其进行专门培训。

(3) 某项目的 SQA 人员不能是该项目组的开发人员、配置管理人员或测试人员,一个项目的 SQA 除了监控项目过程、完成 SQA 相关工作以外,不应该参与项目组的其他实质性工作,否则他会与项目组捆绑在一起,很难保持客观性。

(4) SQA 人员在工作过程中一定要抓住问题的重点与本质,不要陷入对细节的争论之中。SQA 人员应集中审查定义的软件过程是否得到了实现,及时纠正那些疏漏或执行得不完全的步骤,以此来保证软件产品的质量。

(5) SQA 人员应客观、有责任心。作为第三方对项目过程进行监督,应能保持自己的客观性;对于项目组中多次协调解决不了的问题,应向项目的高层管理者反映。

(6) SQA 人员要能应对繁杂的工作。作为 SQA 人员,在跟踪项目进行过程的时候需要对项目组的很多产品进行审计,而且还会参与项目组中的多种活动。同时,一个 SQA 人员还有可能参与多个项目组的审计任务。这就要求 SQA 人员在处理这些事物时要耐心细致。

10.5.4 软件质量保证与测试的区别

目前,很多人对于 SQA 与软件测试的区别不很清楚。在阐述两者区别之前,先介绍 QA 和 QC 的概念。

QA(Quality Assurance,质量保证)是要监控公司质量保证体系的运行状况,审计项目的实际执行情况和公司规范之间的差异,并出具改进建议和统计分析报告,对公司的质量保证体系的质量负责。

QC(Quality Control,质量控制)是对每一个阶段或者关键点的产出物(工件)进行检测,评估产出物是否符合预计的质量要求,对产出物的质量负责。QC 有时也被称为质量检验或质量检查。

QA 在软件企业中实际上就是 SQA,即软件质量保证;而 QC 在软件企业中实际上就是 SQC,即软件质量控制。

如果将软件的生产比喻成一条产品加工生产线,那么 SQA 只负责生产线本身的质量,而不管生产线中单个产品的实际质量情况。也就是说,SQA 只负责软件开发过程的质量,通过保证过程的质量来间接保证软件产品的质量。SQA 在软件企业内对应的角色为软件质量保证人员。

SQC 则不管生产线本身的质量,而只关注按现有生产线生产的阶段性产品的质量是否符合预期的要求。即 SQC 只负责检查软件开发过程中各个阶段产出的工件的质量,如需求规格说明书、设计规格说明书、源代码、测试文档等工件的质量。SQC 在软件企业内对应的角色为软件测试人员。

显然,SQA 和 SQC 的最终目标是一致的,都是为了保证软件的质量。但其工作内容是有很大差别的,SQA 通过控制过程的质量来保证软件产品的质量,而 SQC 则是通过控制整个过程中的阶段性成果的质量来保证软件产品的质量。

SQA 和 SQC 对于企业软件质量的保证是缺一不可的,这两类人员常需要相互配合。

例如，SQC人员（可简称为SQC或QC）在工作过程中会产生出大量的过程数据，SQA人员（可简称为SQA或QA）通过对这些数据进行统计分析，发现软件过程中存在的问题，进而反馈到过程的改进活动中，再通过SQC人员搜集的大量数据来验证软件过程改进的有效性，最终实现过程质量及软件质量的持续改进。

综上所述，SQA与SQC之间的关系如下。

(1) SQA和SQC的共同点在于它们的最终目标一致，都是提高软件的质量。

(2) 二者的区别在于：

- 任务不同。SQA是审计软件过程的质量，而SQC则是检验每个阶段产品的质量。
- 层次不同。SQA是站在比SQC更高的层次上保证软件质量；SQC关注的内容是局部的，而SQA关注的则是整个软件过程。因此，SQC人员（软件测试人员）的工作是受SQA人员的监督的，即SQA可保证测试工作是按照定义好的流程执行的。

10.6 小结

本章介绍了与软件质量及软件测试密切相关的若干概念或活动，如软件质量、软件质量模型、软件度量、软件能力成熟度模型及能力成熟度模型集成、ISO 9000标准、软件质量保证(SQA)等。本章的主要目的是通过对以上内容的介绍，使大家能站在更高的角度看待和把握软件测试、提高软件质量。

习题

1. 软件质量的含义是什么？
2. 软件质量模型有何作用？试列举著名的软件质量模型。
3. 软件度量的含义和作用分别是什么？软件度量的3个主要方面是什么？
4. 什么是CMM？简述其作用。
5. 试简要地解释CMM中5个等级的各自含义。
6. 结合图10-5，说明CMM逻辑结构的含义。
7. CMMI与CMM的主要不同之处在哪些方面？
8. 软件质量保证的目的是什么？其工作内容包括哪些？
9. 软件质量保证与测试的不同之处在哪里？

第 11 章

软件测试案例

本章要点：

- 自动化测试方案选型。
- 数据库系统的压力测试。
- 分布式金融业务系统的性能测试。

本章介绍了三个具体的测试案例，包括一个企业自动化测试方案选型案例和两个性能测试案例。通过对本章的学习，可以强化对软件测试尤其是性能测试的综合体验。

11.1 企业自动化测试方案选型案例

11.1.1 公司背景介绍

A 公司是一家大型保险公司，拥有近 20 个城市的分公司，并在其中 5 个城市建立了 IT 支持中心。这些 IT 中心负责所有内部应用系统的开发和运行维护，同时负责管理 IT 集成商和第三方应用供应商。平均每年的上线应用数量为 20 个左右(新业务系统和原有业务系统的主要版本发布)。其开发团队主要分布在 2 个城市，有 300 名左右，同时 20% 左右的项目通过项目开发外包，或者直接从第三方采购获得。

目前 A 公司的专职测试团队人数不足 30 人，而且测试人员技能参差不齐，几乎没有成熟稳定的自动化测试解决方案。测试只是作为项目上线前的一道工序，并没有发挥应有的作用。但在测试团队和研发部门的沟通等方面，都有明确的邮件往来规范和例会等既定管理方式。由于已上线应用系统的问题，开发团队必须分出一部分资源维护和修复上线应用，而测试团队的工作水平和效率却无法与这些应用质量挂钩，更无从谈起对软件质量的控制。

因此，A 公司决定在软件测试方面进行投入，主要考虑如下几个方面：

- 引进软件测试流程管理的自动化工具，使软件测试和软件开发一样可被评估、被衡量。
- 实现性能测试自动化，所有应用上线之前必须有应用性能风险评估报告和相关部门的确认。

- 逐步实现功能测试自动化，在目前人员配置的情况下，把部分手工测试变成自动化测试。
- 通过软件测试自动化，管理软件测试中的案例、缺陷、报告等资产，进一步提升软件测试效率并建立测试基础库。
- 未来的 2～3 年内使所有应用系统上线都必须有数字化测试数据作为依据。

11.1.2 公司应用系统现状

保险公司的业务种类繁多，同时在经过了几十年的经营后，公司内的应用系统从早期的终端方式到现代的 J2EE 和.NET 等应有尽有。IT 部门计划在未来的 3 年时间内将所有终端和 C/S 方式的应用转换成 B/S 架构，但当前仍然需要对这些旧应用系统进行维护，以保证业务的顺利进行。对于开发部门来说，目前新应用开发基本上已经以 B/S 架构为主，主要是基于 J2EE 架构的 Web HTTP 应用和部分 Window.NET Form 的应用。

11.1.3 公司软件测试现状

测试部门目前仅负责系统测试和对用户验证测试进行管理，对于单元测试和集成测试主要由开发人员完成。由于缺乏监测手段，测试部门无法收集和确定集成测试和单元测试的完成情况。在整个软件测试过程中，业务需求是由开发部门通过 Rational RequisitePro 进行管理，但测试需求尚没有提出要求。测试案例主要放在公司公用的文件服务器的目录中进行管理，对测试中缺陷流程等管理主要依靠邮件的流转进行处理。目前，90％以上的测试是通过 Excel 和 Word 等测试案例文件来完成，测试人员对软件测试自动化的认识仅停留在“记录＋回放”上。

11.1.4 可供选择的方案

A 公司综合考虑了各种软件测试自动化方案，最终把注意力放在以下 3 种方案上。

方案 A 采用以 HP Mercury 公司产品为主的软件测试自动化方案。

(1) 依照原先的邮件流转过程配置 TestDirector 缺陷管理流程，为每个保险业务的开发小组和测试团队分配相应的用户许可证，取消原有邮件方式。

(2) 部署 Mercury QuickTest Pro，完成对应用程序相关功能的测试。

(3) 部署 Mercury LoadRunner，从测试团队中分化出专职的自动化性能测试小组，与业务部门协调，建立 A 公司应用系统上线性能指标，通过 LoadRunner 给出测试指标。

(4) 公司成立专门的质量控制部门，对 TestDirector 中的数据定期进行分析，建立相关质量模型，以便于企业量化管理和过程改进。

方案 B 采用以 IBM Rational 产品为主的软件测试自动化方案。

(1) 采用 Rational TestManager 管理整个测试流程，为相关开发和测试小组成员分配相应权限，改变以前通过邮件和 Word、Excel 文件管理测试的工作方式。

(2) 部署 Rational Robot，以完成相关的功能测试、性能测试，以及新版本发布时的冒烟测试。

(3) 部署 Rational PurifyPlus，将测试工作前移到开发阶段，使编程人员在编码阶段引入的错误能够尽早被检测并修复，降低后期测试的开销。

(4) 公司成立专门的质量控制部门，对 TestManager 中的数据定期进行分析，建立相

关质量模型，以便于企业量化管理和过程改进。

方案 C　采用以开源软件为主的软件测试自动化方案。

(1) 采用 Bugzilla 进行缺陷跟踪管理，采用 Bugzilla Test Runner 进行测试用例管理，采用 CVS 进行测试资源的配置管理。

(2) 采用 MaxQ 和 WebInject 对 B/S 结构的应用系统进行功能测试。

(3) 采用 DBMonster，OpenSTA，LoadSim 进行相关的性能测试。

(4) 可采用 XUnit 架构的开源工具对不同语言的程序单元进行单元测试。

(5) 公司成立专门的开源软件维护小组，以解决遇到的工具维护工作。

(6) 公司成立专门的质量控制部门，对 Bugzilla，Bugzilla Test Runner，CVS 中的数据定期进行分析，建立相关质量模型，以便于企业量化管理和过程改进。

11.1.5　最终采用的测试自动化方案

通过多轮评估和论证，以及经过各厂商的现场演示，甚至包括为期 1 周左右的实际试用，A 公司最终采用了一个以 HP Mercury 公司产品为主的软件测试自动化方案。在这个软件测试自动化方案中，同时考虑了集成 A 公司现有的资源和流程，尽量降低解决方案导入的初期门槛，同时根据要求在实施过程中并非一步到位，而是通过制定详细的实施计划，分步骤展开实施。

1. 初期实施阶段

(1) 部署 Mercury Quality Center 服务器。依照原先的邮件流转过程配置其 TestDirector 的缺陷管理流程，为每个保险业务的开发小组和测试团队分配相应的用户许可证，首先要求所有正在测试和将要进行测试的项目都必须通过这一缺陷管理流程进行缺陷递交和处理，原有邮件方式取消；并安排和培训一名测试管理工程师负责接受测试流程的改进意见和对流程进行优化，以逐步完善流程管理自动化。

(2) 部署 Mercury QuickTest Professional，完成对应用程序的相关功能测试。

2. 中期实施阶段

(1) 通过 Mercury 集成 Rational 的需求管理工具 RequisitePro，实现测试需求的管理。并利用现有的 Mercury Quality Center 平台引入测试案例管理流程，把所有基于 Word 和 Excel 的旧有案例，利用 Mercury 提供的转换工具导入到 TestDirector 中，从而建立可供参考和追踪的测试案例库。同时，由测试部门协调整个执行过程，对测试环境、测试数据、被测系统(SED)的调配实现统一管理，避免可能存在的各部门竞争测试资源的状况。

(2) 部署 Mercury LoadRunner。从测试团队中分化出专职的自动化性能测试小组。通过与业务部门协调，建立 A 公司应用系统上线性能指标。值得注意的是，基于 A 公司应用系统的地域分布性，在通过公司内网远程执行 LoadRunner 测试案例时受到现有网络带宽的限制，很难达到测试效果。尽管 HP Mercury 公司建议 A 公司引进 Mercury Performance Center 平台以实现整合和远程操控，但由于公司预算和目前对性能测试的技能尚不足以建立整合平台，A 公司最终没有采纳该建议。最后，由于性能测试通常是在业务应用经过了一轮的系统测试以后进行的，A 公司决定把性能测试实验室环境集中

在一处，并将需要进行性能测试的应用发布到该实验室进行统一测试。

(3) 对已经开始的功能测试自动化进行优化，从测试小组中筛选出自动化测试专家，负责 A 公司自动化测试库的建立，进一步提升软件测试自动化的价值。

3. 长期实施阶段

(1) 由于自动化测试管理的进行，测试缺陷和测试案例已被大量收集，过程也正在逐步优化。A 公司考虑在 Quality Center 平台上实现“软件质量门户”的思想，即与公司应用软件质量相关的信息都可以通过该平台来得到，相关流程都可以通过这个平台实现，包括作为一个质量门户必须提供的针对软件质量管理和软件测试过程中不同人员角色所定义的视图，衡量软件质量和测试流程效率的关键绩效指标，除测试过程管理以外的服务支持管理(如变更管理、配置管理、发布管理)等。

(2) 考虑整合 A 公司的测试团队和其他分部的非全职测试人员，整合各个团队的测试经验和资源，建立独立的软件质量管理和测试中心。

11.2 SQL Server 2000 压力测试

11.2.1 测试项目概述

2000 年，Microsoft 公司推出了新一代关系数据库系统 SQL Server 2000。用户纷纷对其在数量级或容量级的数据量下的性能表现寄予很高的期望。SQL Server 2000 被推出后不久，某公司对其进行了压力测试，测试它在大记录量和大容量情况下的性能。

11.2.2 测试计划

整个测试过程分为两个部分：第一部分是数据库大容量状态下的执行情况，第二部分是数据库在大记录量下的执行情况。为了方便测试，编写了一个程序进行各种数据库操作，并进行效率记录。

在两个部分的测试中，分别在空数据库环境下进行各种数据库基本操作，并记录各个操作所需要的时间，然后插入了大容量/大记录量的数据后，再进行同样的操作并记录操作所需时间。最后对前后的时间进行比较。由于网络传输等问题，可能导致一些误差，但这对测试不会有太大的影响。

11.2.3 测试准备

1. 测试环境

本次测试的测试环境为：

OS：Windows 2000 Server；

Database：Microsoft SQL Server 2000；

Database Server：ADV 2000。

2. 创建数据库

使用“企业管理器”在数据库服务器上创建数据库 test，并且设置其大小为 10GB，以避免在默认容量下，随着数据库容量增加而导致服务器动态分配磁盘空间的时候引起

开销。

随后再在 test 数据库上创建一个 Tab 表，包含字段如表 11-1 所示。

表 11-1　Tab 表中的字段

字段名	数据类型	可否为空
id(主键)	Int，4	不可
name	Char，10	不可
age	Int，4	不可
photo	Image，16	可

测试人员还使用 Delphi 6 编写了一个测试程序，采用 ADO 接口，连接数据库服务器 ADV 2000。测试程序主要完成以下功能：

- 插入 2000 条数据(Insert)；
- 选择 1000 条数据(Select)；
- 更新 1000 条数据(Update)；
- 删除 1000 条数据(Delete)；
- 插入 100000 条带图片数据(用于大容量测试)；
- 插入 1000000 条不带图片数据(用于大记录量测试)。

11.2.4　测试过程

整个测试过程分为大容量数据测试和大记录量数据测试。

1. 大容量数据测试

在大容量数据测试中，通过插入图片来使数据库的容量膨胀。在以下的所有数据库操作中，都是对带图片的数据进行操作。测试中选择了一张 41958B 的图片，大容量测试是在插入 100000 条记录以后的测试，因此可以大致估计当时的数据表的容量为(41958×100000)/(1024×1024)=4001.43MB。

被测程序首先按顺序执行如下测试：在空表中，插入(Insert)2000 条记录→选择(Select)1000 条记录→更新(Update)1000 条记录→删除(Delete)1000 条记录，并记录下各操作所需要的时间。测试结果如表 11-2 所示。

表 11-2　测试结果(1)

记录数/条	Insert 2000	Select 1000	Update 1000	Delete 1000
操作时间/s	132.781	41.94	0.841	1.552

以上测试是在空数据表中进行数据库的各种基本操作，接下来测试人员插入 100000 条带有图片的记录，使数据表的数据量膨胀到 4001.43MB，以测试大容量环境下的各种数据库操作情况。

同样按照如下步骤进行测试：插入 2000 条记录→选择 1000 条记录→更新 1000 条记录→删除 1000 条记录，并记录下各操作的时间。测试结果如表 11-3 所示。

表 11-3 测试结果(2)

记录数/条	Insert 2000	Select 1000	Update 1000	Delete 1000
操作时间/s	139.05	42.36	0.971	2.264

比较两种环境下的测试结果。表 11-4 列出的是在大容量数据环境下,同样操作增加的时间与初始数据库环境下所需时间的百分比。可以看出,Update 和 Delete 操作在大容量数据环境下的性能损失严重。但是考虑到,因为存取的数据库服务器与客户端位于不同的机器上,虽然通过 100MB 的内部网连接,但是由于网络的问题,导致了存取中网络的延时。当操作所需要的时间比较大的时候,这一点延时对时间比例影响不大,而一旦操作所需要的时间比较短的时候,这个延时就显得尤为突出。

对于 Delete 操作中的 45.88%的性能损失,还有一个原因是,当删除记录的时候,数据库会重建索引。当数据量太大时,导致重建索引耗费了较多的时间。

表 11-4 两种环境下的测试结果比较

记录数/条	Insert 2000	Select 1000	Update 1000	Delete 1000
时间增加百分比/%	4.72	1.001	15.46	45.88

2. 大记录量数据测试

为了让大容量数据测试不影响大记录量数据测试,首先彻底删除大容量测试中使用的表 Tab,并且重新创建一个完全一样的新表 Tab 用于大记录量数据测试。

该测试的步骤与大容量数据测试基本一样,同样测试了空表状况下各个基本数据库操作所需要的时间,但测试人员没有在这个测试中插入图片,photo 字段为空。按以下步骤进行测试:插入 2000 条记录→选择 1000 条记录→更新 1000 条记录→删除 1000 条记录,并记录下各操作的时间,测试结果如表 11-5 所示。

表 11-5 测试结果(3)

记录数/条	Insert 2000	Select 1000	Update 1000	Delete 1000
操作时间/s	16.274	0.07	0.04	0.04

在大容量数据环境下,插入的是 100000 条带有图片的数据,数据量达到 4001.43MB,而在该大记录量环境下的测试,100000 条数据量已经不能满足测试人员的需要,测试人员希望能在更高数据量的环境下进行测试,以便增大加入大数量级数据前后操作效率的差异,因此选择插入 1000000 条数据。

完成了大数量级的数据插入后,再按如下顺序进行测试:插入 2000 条记录→选择 1000 条记录→更新 1000 条记录→删除 1000 条记录,记录下每一个操作所需要的时间。测试结果如表 11-6 所示。

表 11-6 测试结果(4)

记录数/条	Insert 2000	Select 1000	Update 1000	Delete 1000
操作时间/s	16.574	0.05	0.05	0.051

比较两种环境下的测试结果，表 11-7 表示的是在大记录量数据环境下，同样操作增加的时间与初始数据库环境下所需时间的百分比。

表 11-7　两种环境下的测试结果比较

记录数/条	Insert 2000	Select 1000	Update 1000	Delete 1000
时间增加百分比/%	1.84	−28.6	25	27.5

从表 11-7 可以看出 Insert，Update，Delete 操作在大记录量数据环境下性能有所损失，但测试人员无法判断是否是因为这些操作的操作时间太短，还是由于网络引起的误差。

11.2.5　测试结果

虽然在测试中，很多 SQL 操作因为所需时间过短而导致受到网络传输的影响。但是，测试人员仍可通过所需时间较长的 SQL 操作对 SQL Server 2000 的性能进行如下总结。

无论是在大容量(数 GB)还是大记录量(百万条记录量)环境下，SQL Server 2000 的性能都能保持较高的水平。一般情况下的性能损失小于 5%，因此完全能够满足企业通常的应用。

但由于硬件等条件的限制，无法对更大容量(10^1GB，10^2GB 乃至 TB 容量级)及更大记录量(10^8量级记录量)环境下 SQL Server 2000 的性能进行测试。

总体说来，在普通的企业级应用中，SQL Server 2000 已经能够满足应用要求。

11.3　某金融业务系统的性能测试

11.3.1　测试项目概述

被测系统是一个运行在城域网上的大型分布式金融业务系统。其遵循 J2EE 规范，采用 B/S 体系结构进行设计和开发。处理的业务主要分为交易业务和查询业务。系统体系结构如图 11-1 所示。体系结构中的各部分解释如下。

(1) 表示层

运行在终端上。运行 Java Applet 程序，提供协议控制和用户界面，与系统最终用户实现直接交互，通过 TCP/HTTP 与前置系统通信。向前置系统发送请求报文，并接收前置系统返回的回应报文。

(2) 商业逻辑层

商业逻辑层作为中间层实现核心业务逻辑服务，包括交易应用服务、交易前置服务和查询前置服务。

其中，交易应用服务运行在交易主机上。在 Tuxedo 中间件上运行业务处理程序，按交易规则处理前置机发来的交易指令，通过 Tuxedo Jolt 与前置机连接，通过 DB2 C API 与数据库连接。

交易前置服务和查询前置服务运行在前置机上。交易前置服务运行服务程序接收终端请求报文并通过 Tuxedo Jolt 客户端将其转发给交易主机，再通过轮询和同步反馈接

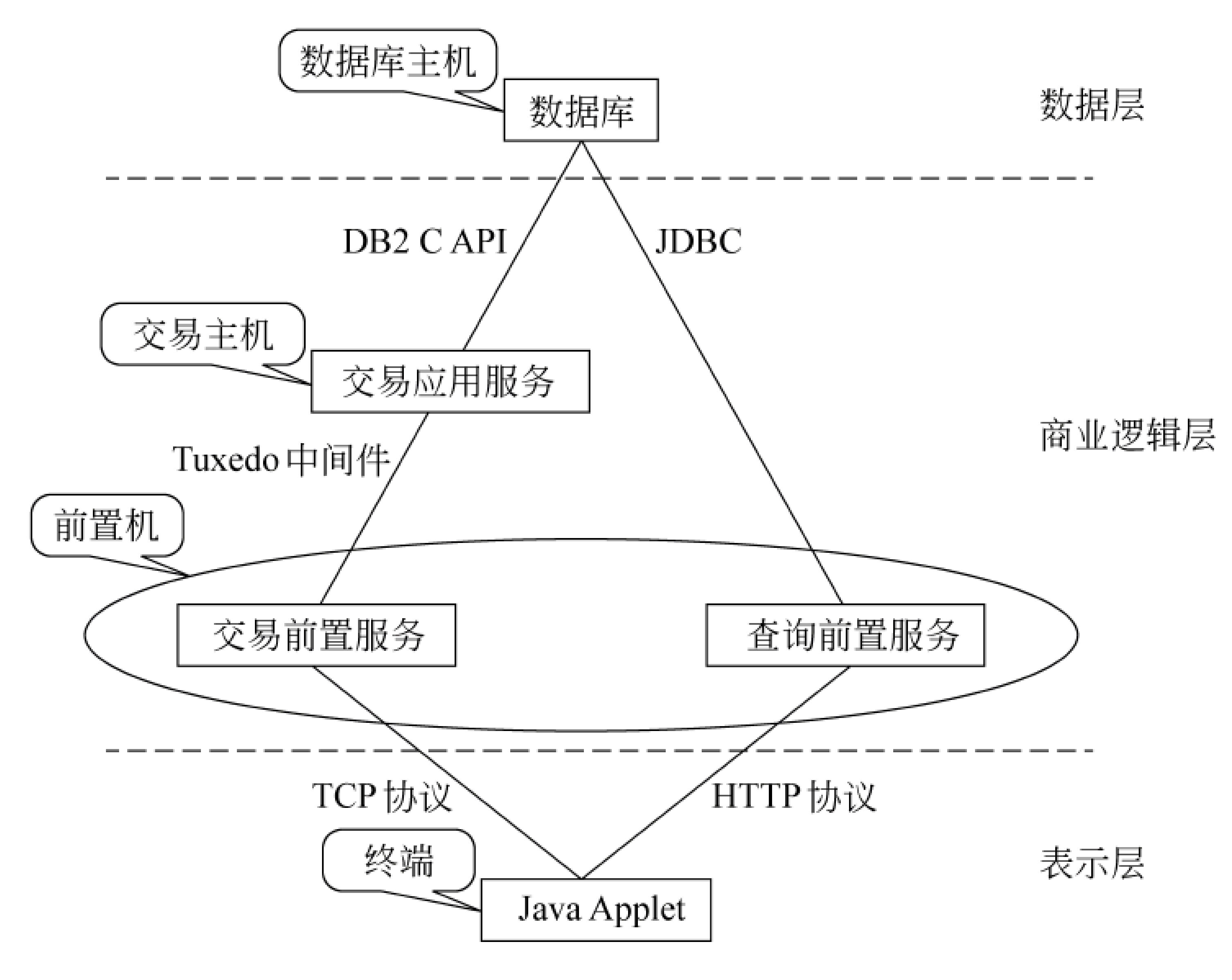


图 11-1 被测系统体系结构图

收交易主机返回的报文，将其转发给业务终端；查询前置服务运行在 WebLogic 应用服务器上并调用 JReport 组件，通过 JDBC 完成对查询流指令的发送并接受数据库返回的结果给业务终端。

（3）数据层

数据层运行在数据库主机上。负责整个系统中数据信息的存储、访问及其优化。数据层运行 DB2 数据库服务程序，通过 DB2 C API 与交易主机通信，并通过 JDBC 与查询前置服务通信。数据库主机和交易主机运行在交易中心城市，前置机运行在各个分中心城市，终端为各个城市参加交易的单位，整个系统覆盖城域网。

11.3.2 被测系统的性能要求

金融系统是业务处理十分频繁、数据吞吐量极大的系统，业务处理的速度将直接关系到公司的经济效益和客户对公司的评价。在客观条件下，系统必须在大业务量情况下同时保持快速的实时响应能力，以保证整个业务系统的通畅运行。用户对该系统提出的性能要求如表 11-8 所示。用户的性能要求是进行性能测试的依据。

表 11-8 用户要求的性能指标

指标种类		用 户 需 求
登录		系统能够处理 750 用户/min，至少支持上百用户并发，登录响应时间不超过 30s
业务	交易	系统能够处理 300 笔/min，响应时间不超过 10s
	查询	系统能够处理 400 笔/min，响应时间不超过 10s

11.3.3　性能测试过程

1. **对被测系统进行系统分析**

被测系统大体上由终端、前置机、交易主机、数据库主机节点组成。在整个业务流程中，业务终端→前置机→交易主机→数据库主机形成了一个压力流串，每个节点在压力下能够正常工作是整个系统正常运转的基础。也就是说，如果其中任意一个节点在业务压力下发生了拥塞、处理不力等不正常情况，则整个系统都无法正常运转。

系统的业务流程如下。

首先，从终端到前置机，终端产生业务报文发送至前置机，前置机上运行查询前置服务和交易前置服务，查询前置服务向下通过 HTTP 协议以 Web 服务形式和终端连接，向上通过 JDBC 直接与数据库系统相连。交易前置服务向下通过基于 TCP 协议的 Socket 连接和终端通信，向上通过 Tuxedo Jolt 客户端和交易应用服务连接。交易应用服务进行业务逻辑计算，并操作数据库系统。

由上述分析可以整理出整个系统的两条压力流程线，之所以分为两条流程线，是因为交易前置服务和查询前置服务的工作原理完全不同，下与终端的连接，上与交易主机的连接也是完全独立的两个通路，即：

① 终端→交易前置机→交易主机→数据库系统。

② 终端→查询前置机→数据库系统。

下面先独立分析两条流程线，再综合分析二者之间的相互影响。

第一条路线上主要运行的是登录指令和交易指令信息。

当系统运作时，多个交易终端与交易前置服务建立 Socket 连接，完成登录，之后发送交易指令，造成对交易前置服务的压力。交易前置服务通过运行服务程序接收到交易指令，并检验其合法性，然后通过交易中间件 Tuxedo 的客户端把业务的压力传递给交易主机进行处理。交易主机进行必要的金融计算和业务逻辑运行，得出反馈结果，生成消息，一方面顺原路返回到各个终端上去，一方面记录到数据库中。

在本条流程线上的加压主要考验交易前置服务程序的 Socket 多连接建立能力，Tuxedo 交易中间件的即时响应能力，交易主机的计算能力，以及 DB2 数据库的 DML 语句加锁机制。

第二条路线上主要运行的是查询指令信息。

查询指令产生时，通过 HTTP 协议访问 WebLogic 上的 Web 服务器和应用服务器上的相应组件，以 JDBC 接口访问后台的 DB2 数据库，并把数据库返回的结果发送至终端界面。在本条流程线上的加压主要验证 WebLogic 处理能力，以及数据库中索引是否创建合理。

两条流程线相对独立，但又是互相依赖的。由于是对同一个数据库系统进行读操作和写操作，查询流程的结果依赖于交易流程数据的产生，交易流程产生的数据又通过查询流程得到验证。在进行压力测试时，两者的协同会对数据库形成压力的冲击。

由以上分析，结合用户要求的性能指标，决定把本次性能测试分解为如下两种子测试进行。

① 并发登录测试：750个终端1分钟内并发登录系统，且响应时间应在30秒之内。

② 业务负载测试。

业务负载测试含有3个子测试如下：

① 交易流程测试。多个终端发起交易请求，逐渐加压，以达到300笔/s的压力为限。

② 查询流程测试。多个终端进行查询，逐渐加压，以达到400笔/s的压力为限。查询成功与否以所请求的Web页面完全展现为标准。实际上，查询响应能力与数据库中的数据量有关系，后来经与用户进一步确认，基础数据为30万条。

③ 综合测试。在上面两种测试都通过的情况下，进行综合测试。

2. 性能测试的执行过程

对被测系统进行系统分析，确立了测试方案后，本性能测试依照以下的步骤执行。

(1) 测试脚本的开发

本性能测试利用HP Mercury公司的LoadRunner进行，脚本编辑和编译工作在Virtual User Generator中完成。

对于并发登录测试和交易流程测试，两者运行机理相同，都是终端调用Socket client，和交易前置的Socket server建立连接，将请求消息发送至交易前置机。故将此部分Java Socket程序编入测试脚本程序，生成登录和交易业务脚本，通过LoadRunner执行。这样做的好处是绕过终端IE界面复杂的处理逻辑，直接施压在前置机上。

脚本除了需要实现与前置机的Socket连接、业务发送等功能，还要建立用户信息数据池，设置检测点、异常退出点，为脚本执行后的结果统计和分析提供正确的依据。

交易业务脚本的核心内容如下。

```
public class Actions
{
    /* 登录变量初始化 */
    ProtocolManager protocol;                    //ProtocolManager 为实现 Socket 连接的类
    ServiceName service;                         //ServiceName 对服务端的信息进行了封装，包括
                                                 //IP 地址和端口号
    LoginMessage login;                          //LoginMessage 为登录时需要向服务器发送的消
                                                 //息，待服务器确认并返回回应消息时登录成功
    protocol=new ProtocolManager();              //创建 ProtocolManager 类 protocol 对象
    service=ServiceName.getInstance();           //获得 ServiceName 实例
    login=new LoginMessage();                    //创建 LoginMessage 类 login 对象
    service.setIP("202.31.10.18");               //设置服务端 IP 地址
    service.setPort(17777);                      //设置服务端端口号
    /* 设置登录消息 */
    login.setUserName(lr.eval.string("{loginName}"));    //从数据池里读出用户名，设置在
                                                 //login 成员变量里
    login.setPasswd("1234");                     //数据库中添加的用户密码都为 1234
    /* 发送登录消息 */
    protocol.login(login);                       //发送登录消息
    lr_start_transaction("trade");               //交易开始点
    TradeMessage trademessage;                   //生成交易消息
    /* 设置交易消息 */
```

```
    …
  /* 发送交易消息 */
    …
  if(sendfail)
    lr_end_transaction("trade", LR_FAIL);//如果发送交易消息失败,交易结束,返回
  /* 循环回收主机返回的处理信息 */
    …
  if(recievefail)
    lr_end_transaction("trade", LR_FAIL);//如果不能接收到主机处理回应消息,交易结束,
                                         //返回
  if(recievesuccess)
    lr_end_transaction("trade", LR_PASS);  //如果接收到主机成功处理的回应消息,交易结
                                           //束,返回
    …
}
```

在上面的例子中,主要对每笔交易进行了 Transaction 化。在交易开始时设置开始检测点,交易结束时设置结束检测点,并向 LoadRunner 报出交易状态。实际的脚本中在回收交易响应消息时还进行了拆包,在应用层上对交易状态进行识别,并非例子中只在 Socket 层加以判断。

对于查询流程测试,由于 LoadRunner 工具支持基于 HTTP 的 Web 访问录制功能,故采用录制脚本为主,手工编写脚本为辅的方法,生成查询业务脚本,通过 LoadRunner 来执行。

(2) 根据用户的性能要求创立测试场景

在本次性能测试中,用户提出的性能指标不够细致和确切,测试人员通过对用户调查和实际业务分析,将性能指标的实现方式进行了明确的定位。

① 并发登录测试场景。并发登录 750 用户/min,登录响应时间在 30s 之内。这里的并发登录 750 用户/min 指的是系统能够在 1min 内接受 750 个用户的登录请求,而处理的效果如何则在交易终端体现,即登录响应时间。基于这样的理解,把用户性能指标转化为如下的测试场景:

从第一秒钟开始,用 LoadRunner 每秒钟登录 13 个用户,并保持 Socket 连接,直到 1 分钟结束,从终端向系统一共发送 750 个左右的用户登录请求。在终端观察并统计登录响应时间。如果系统不能响应持续增加的登录请求或平均登录响应时间大于 30s,则并发登录测试场景不能通过。

为了帮助用户更加深入了解系统的能力,还对系统的瞬时并发能力进行了测试,即测试系统所能承受的最大的瞬时并发用户登录连接请求个数。这个场景通过 LoadRunner 在登录前设置同步点来实现,这个结果将结合上一个结果共同反映系统的登录处理能力。

② 交易流程和查询流程测试场景。在这里只对系统的业务负载能力做测试(并发处理能力在登录测试中已得到验证)。测试场景如下:在 LoadRunner 中,建立 goal-oriented 的测试场景,以 400 笔/s 为目标,将调度权交给 LoadRunner 来试图达到这个指标。

③ 综合测试场景。交易流程测试和查询流程测试同时进行。

以上的测试场景要求均可在 LoadRunner 中的 Controller 进行设置完成。

(3) 运行测试场景,同步监测被测系统性能

在 LoadRunner 中的 Controller 中开启 UNIX 系统资源计数器、WebLogic 计数器、DB2 计数器,检测系统资源消耗情况,并最终与和测试结果数据合并,成为分析图表。

测试结果可在测试执行完毕,通过 LoadRunner 中的 Analysis(分析器)获得。

- 并发登录测试。依照设计好的测试场景,用 LoadRunner 工具在 1min 内渐增地向系统发送登录请求。分别进行 3 次,结果如表 11-9 所示。

表 11-9　登录测试结果

测 试 序 列	登录成功的用户数	平均响应时间(s/笔)
1	485	25.6
2	497	27.3
3	482	26.2

这里的登录成功用户指的是系统接受了登录请求,并建立了连接。为获取平均响应时间,可在登录脚本里设置检测点,由 LoadRunner 工具自动获得。

为考查系统的瞬时并发处理能力,在完成上一步测试的前提下,逐步增加瞬时并发登录用户数,直到系统极限。测试执行结果如表 11-10 所示。

表 11-10　瞬时并发登录测试结果

瞬时并发用户数	计划并发数	1	2	10	100	200
	实际并发数	1	2	10	100	122
用户登录响应时间	最短/s	2.0	3.1	7.2	21.2	24.2
	平均/s	2.0	3.3	12.2	24.3	28.1
	最长/s	2.0	3.5	16.5	31.2	33.3

- 负载测试。

① 交易流程测试。交易流程测试结果如下。

对于通过网络接口发送的批量业务请求,均在性能指标所指定的时间范围内得到请求成功的反馈消息,说明主机已经处理成功。

在通过网络接口发送业务请求的同时,开启 Internet Explorer,通过实际终端界面进行登录和交易,系统响应时间延长,界面显示和刷新明显变慢,到业务量高峰时期,界面已经不能显示任何信息,处于不可工作的状态。

需要说明的是,系统正常工作时,每个界面终端不仅应该能够展示己方的交易信息,还要展示其他交易单位的交易信息和系统信息。因此当交易量大的时候,界面需要展示的信息量是巨大的,这本身对终端界面是一个性能考验。

② 查询流程测试。本流程测试在交易流程测试之后进行,以利用其生成的数据。测试结果基本满足性能指标。

③ 综合测试。由于交易流程测试未通过,本测试已经不能执行。

(4) 测试结果分析及性能评价

① 并发登录测试结果分析。根据表 11-9 和表 11-10 可以得出以下结论:

- 被测系统在一分钟内并不能接受 750 个用户的登录请求,其可接受的登录请求用户数约为 490 个。在这样的条件下,登录响应时间在用户要求范围之内。
- 被测系统的瞬时并发处理能力约为 122 个用户。

② 交易流程测试结果分析。根据交易流程测试结果可知,通过脚本程序进行业务行为,发送业务请求消息到回收主机处理回应消息,这段时间系统是顺畅的,反应也是迅速的,但是在终端界面却不能即时展现消息。这说明信息的回馈通路在终端界面出现了性能瓶颈。当界面需要在短时间内展示大量交易信息时,已经不能承受负荷。这与终端采用 Java Applet 技术有关。

③ 查询流程测试结果分析。查询流程基本符合性能指标。

需要说明的是,实际中,以上每个场景的测试都执行了多次,中间件参数进行了多次的调优。从以上测试的结果分析也可以看出,性能测试瓶颈不是出现在中间件产品上,而是在自身开发的程序上。

11.4 小结

本章对 3 个具体的软件测试案例进行了介绍,其中包括一个企业自动化测试方案选型案例和两个性能测试案例。通过本章的学习,可加强对软件测试的综合体验。但由于项目背景、测试需求、公司偏好的不同,软件测试的案例千变万化。这些案例显然不能囊括软件测试的全部方法和技术,大家应尝试接触更多的测试案例,最好是亲身投入到测试实践中。

附录 A

常见的软件测试术语

英文名称	中文名称	术语简介
Acceptance Testing	验收测试	以用户为主的测试。对整个软件系统进行测试和评审，决定是否接收软件
Actual Outcome	实际结果	被测对象在特定条件下执行时产生的结果
Ad Hoc Testing	随机测试	没有书面测试用例的测试。主要是根据测试者的经验对软件进行功能和性能抽查
Alpha testing	α 测试	软件开发公司内部人员模拟各类用户对即将交付的软件产品进行的测试。一般在可控制的环境下进行
Anomaly	异常	在文档或软件操作中观察到的与期望相违背的结果
Architecture	架构	一个系统或组件的组织结构
ASQ	自动化软件质量	使用软件工具提高软件的质量
Assertion	断言	一个布尔陈述。指明在程序执行期间某个点上程序变量应满足的条件
Audit	审计	对一个或一组工作产品的独立检查，以评价它与规格、标准等的符合程度
Audit Trail	审计跟踪	系统审计活动的时间记录
Automated Testing	自动化测试	使用测试工具进行测试
Baseline	基线	在软件开发中经过正式审核作为下一步开发基础的配置项，是项目情况的度量点及对后继开发工作进行验证的基准
Basic Block	基本块	若干顺序的可执行语句构成的块，不包含任何分支语句
Basic Path Testing	基本路径测试	确定程序的环路复杂性，导出基本路径集合，进而在其基础上设计测试用例，这些测试用例能覆盖程序中每一条可执行语句
Benchmark	基准/标杆	可用于度量和比较的标准
Beta Testing	β 测试	由用户进行的测试。目的在于帮助开发方在正式发布软件产品前对其进行最后的改进。测试环境一般不可控

续表

英 文 名 称	中 文 名 称	术 语 简 介
Big-bang Integration	一次性集成/大爆炸集成	属于非增值式集成测试方式。先分别对每个组件进行测试,然后一次性地将所有组件集成在一起进行测试
Black-box Testing	黑盒测试	不考虑系统或组件的内部实现,根据规格说明对其进行测试
Boundary Value	边界值	处于组件输入等价类或输出等价类边界上的值
Boundary Value Analysis	边界值分析	针对组件的输入和输出等价类边界进行分析,使设计的测试用例能达到边界值覆盖
Boundary Value Coverage	边界值覆盖	测试用例设计方法。通过设计足够多的测试用例,覆盖组件中各输入和输出等价类的所有边界值
Branch	分支	在组件中控制从某语句到其非直接后继语句的转移
Breadth Testing	广度测试	测试一个软件系统或组件的所有功能,但不测试其更细节的特性
Bug	缺陷	在软件系统或组件中一个错误的表现
Capture/Replay Tool	捕获/回放工具	测试工具。能捕获在测试过程中传递给软件的输入,并在以后重复执行捕获的内容
CASE	计算机辅助软件工程	即 Computer Aided Software Engineering。支持软件开发自动化的一套方法和工具
CAST	计算机辅助软件测试	即 Computer Aided Software Testing。通过使用测试工具实现测试过程部分自动化的测试
Cause-effect Graph	因果图	用于描述输入条件取值组合及与每种组合对应的输出的图形,可在其基础上设计测试用例
Change Control	变更控制	用于软件开发过程中,对变更进行管理,确保变更有序地进行
CMM	软件能力成熟度模型	即 Capability Maturity Model for Software。是一个率先在软件行业从软件过程能力的角度提出的软件评估标准,通过它能评价一个软件机构开发过程的能力成熟度
Code Audit	代码审计	由人或工具对源代码进行的独立检查,以评价其与设计规格、编码标准等的一致性
Code Coverage	代码覆盖率	一个测试套执行后,对组件中代码的覆盖程度
Code Inspection	代码审查	正式的同行评审手段。审查小组成员依据缺陷检查表对程序中的逻辑提出疑问,检查它与设计规格和编码规范的一致性
Coding Standards	编码规范	使用某种语言编程时需要遵循的标准
Compatibility Testing	兼容性测试	验证软件能否与不同的软件协作运行的测试
Complexity	复杂性	软件系统或组件难以理解或验证的程度
Component	组件	最小的软件单元,有独立的规格

续表

英 文 名 称	中 文 名 称	术 语 简 介
Component Testing	组件测试	参见单元测试
Computer System Security	计算机系统安全性	计算机软硬件对故意的(或偶然的)破坏的防御能力
Condition	条件	分为简单条件(关系表达式)和复合条件(布尔表达式),取值为真或假
Condition Combination Coverage	条件组合覆盖	设计足够多的测试用例,覆盖被测组件的所有判定中的全部简单条件取值组合
Condition Coverage	条件覆盖	设计足够多的测试用例,使被测组件中的每个简单条件取到各种可能的结果
Configuration Management	配置管理	在软件开发过程中标识、组织和控制变更的技术。目的是使错误降为最小并最有效地提高开发效率
Configuration Testing	配置测试	验证软件能否与相关硬件正常地协作运行的测试
Conformance Testing	一致性测试	用于验证一个系统或组件的实现是否和其规格说明相一致的测试
Control Flow	控制流	程序执行过程中所有可能的事件顺序的一个抽象表示
Control Flow Graph	控制流图	通过一个组件的所有可能控制流路径的图形表示
Conversion Testing	转换测试	一种用于测试已有系统的数据能否转换到替代系统上的测试
Correctness	正确性	软件在其分析、设计、编码中无故障的程度;软件、文档及其他项满足用户显式和隐式需求的程度
Coverage	覆盖率	基于某种覆盖项(如语句、判定取值、条件取值等),测试执行时对覆盖项的覆盖比例
Crash	崩溃	系统或组件突然完全丧失功能
Criticality	关键性	需求、功能实现、故障等对系统操作或开发影响的程度
Criticality Analysis	关键性分析	对需求关键性的分析,用于给每个需求项分配一个关键级别
Cyclomatic Complexity	环路复杂性	其度量值等于程序中的独立路径数目
Data Dictionary	数据字典	存储在目录中的一组系统表,包括对数据库结构和相关信息的定义
Data Flow Coverage	数据流覆盖	测试覆盖率的度量是根据变量在代码中的使用情况
Data Flow Diagram	数据流图	一种用于结构化分析的图形工具,它从数据传递和加工的角度刻画系统内的数据运动情况。有 4 种基本成分:加工、数据流、数据存储和外部实体
Data Flow Testing	数据流测试	根据代码中变量的使用情况进行的测试
Dead Code	死代码	程序中永远不可能被执行到的代码
Debugging	调试	寻找并清除软件失效根源的过程

续表

英文名称	中文名称	术语简介
Decision	判定	程序中的一个控制点,在此处控制流有两个或多个可替换路由
Decision Coverage	判定覆盖	设计足够多的测试用例,使被测组件中的每个判定取到各种可能的结果
Decision Outcome	判定结果	判定的取值,用来决定控制流的走向
Decision Table	判定表	一种用于表示输入条件取值组合及与每一组合对应的输出的图形,可在其基础上设计测试用例
Depth Testing	深度测试	测试软件系统或组件的一个功能的所有特性,但不测试其所有功能
Desktop Checking	桌面检查	编程人员手工模拟执行自己编写的代码并进行分析,以尽可能地发现代码中的错误
Documentation Testing	文档测试	对提交给用户的文档进行的测试,目的是提高文档的可靠性和易用性,降低技术支持费用
Down-Top Testing	自底向上测试	属于增值式集成测试方式。从软件的最底层组件向上逐步集成,在集成的同时进行测试,直至集成为符合要求的最终软件系统
Dynamic Analysis	动态分析	根据执行时的行为评价一个软件系统或组件
Dynamic Testing	动态测试	通过执行软件系统或组件来对其进行测试
Embedded Software	嵌入式软件	运行在特定硬件设备中的一种软件,不直接与用户交互。对其实时性、可靠性要求较高
Entity Relationship Diagram	实体关系图	一种用于描述现实世界中实体及它们之间关系的图形
Equivalence Class	等价类	组件输入或输出域的一个部分,在该部分中,组件的行为从规格上看是相同的
Equivalence Class Partition	等价类划分	一种测试用例设计方法,对某组件划分出等价类,从等价类中选择代表性数据作为测试用例
Error	错误	IEEE 的定义是:一个人为产生不正确结果的行为
Error Guessing	错误推测	一种测试用例设计方法,根据测试人员的经验推测系统或组件中可能出现问题的地方,有针对性地设计测试用例
Exception	异常/例外	一个引起正常程序执行挂起的事件
Executable Statement	可执行语句	一种语句,在被编译后会转换成目标代码,程序运行时会被执行,且可能对程序中的数据产生动作
Exhaustive Testing	穷举测试	对于黑盒测试,测试系统或组件的所有输入条件取值组合;对于白盒测试,测试组件中所有路径
Failure	失效	软件的行为与其期望的行为相背离
Fault	故障	在软件中一个错误的表现
Fault Tolerance Testing	容错性测试	通过构造一些不合理的输入引诱软件出错,检测软件的容错能力

续表

英 文 名 称	中 文 名 称	术 语 简 介
Feasible Path	可达路径	通过设置输入条件取值执行到的路径
Functional Testing	功能测试	也称为行为测试，其测试一个软件系统或组件的特性和可操作行为以确认它们满足需求规格说明中的相关功能需求
Gray-box Testing	灰盒测试	一种介于白盒测试和黑盒测试之间的测试，它基于组件运行的外部表现同时又结合其内部逻辑来设计测试用例。一般认为，集成测试阶段采用的测试策略近似于灰盒测试
High-frequency Integration	高频集成	同步于软件开发过程，每隔一段时间对开发团队的现有代码进行一次集成测试
Incremental Testing	增值式测试	一种集成测试策略，组件逐渐被集成到系统中直至形成完整的软件，在集成的过程中进行测试而非一次性地测试
Infeasible Path	不可达路径	不可能通过任何输入条件取值执行到的路径
Installation Testing	安装测试	对软件系统可安装性的测试
Instrumentation	插桩	在程序中插入额外的代码以获得程序在执行时行为的信息
Instrumenter	插桩器	执行插桩的工具
Integration Testing	集成测试	一般在单元测试之后进行，目的是确保集成到一起的各组件能共同完成预期的功能，并达到要求的性能
Intensity Testing	强度测试	性能测试的一种，目的是找出因应用系统中资源不足或资源争用而导致的错误
Interface Analysis	接口分析	分析软件和硬件、用户及其他软件之间接口的需求规格
Interface Testing	接口测试	对组件之间接口正确性的测试
Invalid Input	无效输入	在组件功能输入域之外的输入数据
Job	工作	一个用户定义要计算机完成的工作单元
Load Testing	负载测试	对系统施加越来越大的负载，综合分析交易执行指标和资源监控指标，评测和评估应用系统在不同负载下的性能，以定位性能瓶颈，优化系统性能
Localization Testing	本地化测试	测试软件本地化版本的质量，测试的内容主要是软件本地化后的界面布局和软件翻译的语言质量
Logic-coverage Testing	逻辑覆盖测试	通过分析程序的内部逻辑结构并考虑对其覆盖程度来设计测试用例的方法
Maintainability	可维护性	软件系统或组件可被修改的容易程度，这个修改一般是由于缺陷纠正、性能改进、特性增加等引起的
Maintainability Testing	可维护性测试	对软件系统或组件可维护性的测试

续表

英 文 名 称	中 文 名 称	术 语 简 介
MTBF	平均无故障时间	即 Mean Time between Failures。两次失效之间的平均操作时间
MTTR	平均修复时间	即 Mean Time to Repair。修理和恢复失效系统的平均时间
Negative Testing	逆向测试/负面测试	针对违反需求规格的情况设计测试用例
Operational Testing	可操作性测试	在软件系统或组件操作的环境中评价其表现
Path	路径	一个组件从入口到出口的由若干可执行语句构成的语句执行序列
PDL	过程设计语言	即 Procedure Design Language。是用来描述模块内部算法设计和加工细节的非正式语言，是一种伪码
Performance Testing	性能测试	评价一个软件系统或组件与性能需求是否符合的测试，包括负载(压力)测试、强度测试、容量测试、疲劳测试等类型
Pilot Testing	引导测试	目的在于在软件开发中验证系统在真实硬件和客户基础上处理典型操作的能力
Portability Testing	可移植性测试	验证软件能否被移植到指定的硬件或软件平台上的测试
Pseudo-random	伪随机	看似随机，实际上是根据预先安排的顺序进行的
QA	质量保证	即 Quality Assurance。监控公司质量保证体系的运行状况，审计项目的实际执行情况和公司规范之间的差异，并出具改进建议和统计分析报告
QC	质量控制	即 Quality Control。对每一个阶段或者关键点的产出物(工件)进行检测，评估产出物是否符合预计的质量要求，对产出物的质量负责
Recoverability Testing	恢复性测试	将软件置于极端的条件下，或者是模拟的极端条件下迫使其产生故障，检测其恢复正常工作状态的能力
Regression Testing	回归测试	为验证对软件引入的修改的正确性及其影响而进行的测试
Reliability	可靠性	在一定的环境下，系统不发生故障的概率
Requirements-based Testing	基于需求的测试	根据软件系统或组件的需求规格导出测试用例的测试设计方法
Review	评审	在软件开发过程中，把阶段性的软件配置提交给开发人员、用户、管理者等进行评价、审批的过程
Risk	风险	不期望效果的可能性和严重性的一个度量
Robustness Testing	健壮性测试	目的在于检测软件在异常情况下能继续正常运行的能力。包括容错性测试和恢复性测试
Sanity Testing	健全测试	参看冒烟测试(Smoke Testing)

续表

英文名称	中文名称	术语简介
SC	软件配置	即 Software Configuration。是软件生存周期各阶段产生的各种形式和各个版本的文档、程序、数据及环境的集合
SDP	软件开发计划	即 Software Development Plan。是指管理软件项目的全面计划
Security	安全性	参看计算机系统安全性(Computer System Security)
Security Testing	安全性测试	验证系统是否达到安全性目标的测试
Simulation	模拟	使用另一个系统来表示一个物理或逻辑的系统的特定行为特性
SLA	服务级别协议	即 Service Level Agreement。服务提供商和客户之间的一种协议,用于规定提供的服务级别
Smoke Testing	冒烟测试	是一种典型的初始测试,它对一个新的软件版本的主要功能成分进行简单的测试,以判断其能否进行后续的正式测试
Software Deployment	软件部署	又称软件发布。指软件的第一个版本通过彻底的测试、形成产品、交付给客户的阶段
Software Engineering	软件工程	是用于分析、设计、实现、与维护软件系统的一组规范和方法,它指导着软件开发人员以工程化的手段规范地开发高质量的软件
Software Life Cycle	软件生命周期	从设想一软件产品开始到该软件被废弃为止的时间间隔
Software Measurement	软件度量	是对软件开发项目、过程及其产品进行数据定义、收集及分析的持续量化过程,以对项目质量、过程质量及产品质量进行理解、预测、评估、控制和改善
Software Quality	软件质量	IEEE 的定义是:系统、部件或过程满足顾客或者用户需要或期望的程度;系统、部件或过程满足规定需求的程度
Software Test Documentation	软件测试文档	主要包括测试计划、测试设计、测试用例、测试规程、测试事件报告、测试总结报告等。它为软件测试项目的组织、规划和管理提供了一个架构
Source Code	源代码	可输入到编译器或其他转换设备上调试运行的代码
Specification	规格	用于定义系统或组件的功能、性能及其他特性的文档
SPP	软件项目计划	参看软件开发计划(SDP)
SQA	软件质量保证	即 Software Quality Assurance。是 CMM 第 2 级中的一个关键过程域,其目的是向管理者提供全面监控软件过程的手段,验证软件产品和活动是否符合相应的规程和标准,并给出统计分析报告和改进建议
SQL	结构化查询语言	即 Structured Query Language。一种用于关系数据库中查询和处理数据的语言

续表

英 文 名 称	中 文 名 称	术 语 简 介
State	状态	软件对象在其生命周期内满足特定条件的存在，在此条件下，对象能执行特定的动作或等待事件的发生
State Diagram	状态图	用来描述一个特定对象的所有可能状态以及引起状态转移的事件，表示单个对象在其生命周期中的全部行为
State Transition Testing	状态转换测试	根据状态间的转换来设计测试用例的方法
Statement	语句	语句是程序的基本单位之一，是程序具体操作内容的体现，一般每条语句以分号结尾
Statement Coverage	语句覆盖	设计足够多的测试用例，使被测程序中的每条语句至少被执行一次
Static Analysis	静态分析	对被测对象的逻辑、语法等进行分析，但并不执行它
Static Testing	静态测试	通过静态分析而非动态执行完成对系统或组件的测试
Stress Testing	压力测试	参看负载测试(Load Testing)
Structural Test Case Design	结构化测试用例设计	参看逻辑覆盖测试(Logic-coverage Testing)
Structural Testing	结构化测试	参看逻辑覆盖测试(Logic-coverage Testing)
Structured Design	结构化设计	一种传统的设计方法，要点是：使用独立功能、单入口、单出口的模块；自顶而下、逐步求精
Structured Programming	结构化编程	依照结构化设计的结果，用结构化编程语言进行的编程
Stub	桩	为一个软件模块的框架，用来在开发或测试中代替当前模块的子模块
Symbolic Evaluation	符号评价	参考符号执行(Symbolic Execution)
Symbolic Execution	符号执行	通过符号表达式来执行程序路径的一种静态分析技术。其中，程序的执行用符号来模拟，例如，使用变量名而不是实际值，程序的输出被表示成包含这些符号的逻辑或数学表达式
Syntax Testing	语法分析	根据语法验证组件的测试用例设计方法
System Integration	系统集成	各组件渐增式地加入到系统中，直至成为一个完整的系统
System Testing	系统测试	将开发出的软件，作为基于整个计算机系统的一个元素，在实际运行环境下或模拟系统运行环境下，测试其与系统中其他元素能否实现正确地连接，以满足需求规格
TDD	测试驱动开发	即 Test-Driven Development。强调通过测试来推动整个开发的进行，即在明确要实现某个功能后，首先编写对该功能的测试代码，接着编写相关的代码满足这些测试代码，然后循环添加其他功能，直到实现全部功能

续表

英 文 名 称	中 文 名 称	术 语 简 介
Test Case	测试用例	为特定目标开发的一组测试输入、执行条件和预期结果，其目的是测试程序中的某路径，或核实程序或软件能否完成某个特定的功能
Test Case Suite	测试用例套	对应于被测软件某功能或特性的多个测试用例的集合
Test Comparator	测试比较器	一种测试工具，用于比较系统或组件执行用例的实际结果和预期结果
Test Completion Criterion	测试完成标准	用于确定被计划的测试何时完成的标准
Test Environment	测试环境	测试用例运行于其上的软件、硬件及网络环境
Test Log	测试日志	关于测试执行的所有相关细节的时间记录
Test Plan	测试计划	一种测试文档，用于描述测试活动的范围、方法、资源和进度
Test Procedure	测试规程	规定对于运行系统和执行指定的测试用例来实现有关测试设计所要求的所有步骤
Test Records	测试记录	对每个测试，明确地记录被测系统或组件的标识、版本、测试规格和执行结果
Test Report	测试报告	一种用于描述系统或组件执行的测试和结果的文档
Test Script	测试脚本	是一个特定测试对应的一系列指令（及数据），这些指令可以被测试工具自动执行
Test Specification	测试规格	一种文档，用于指定对系统或组件的特性或特性组合的测试方法及测试用例
Test Strategy	测试策略	用于描述测试的目标和大致方法
Test Suite	测试套	参看测试用例套（Test Case Suite）
Testability	可测试性	一个系统或组件有利于测试标准建立和验证这些标准是否被满足的程度
Testing	测试	使用人工或自动的手段来运行或测定某个软件系统的过程，其目的在于检验它是否满足规定的需求或弄清预期结果与实际结果之间的差别
Testing Item	测试项	作为测试对象的工作版本
Top-Down Testing	自顶向下测试	属于增值式集成测试方式。从软件的最顶层组件向下逐步集成，并在集成的过程中进行测试，直至集成为符合要求的最终软件系统
UML	统一建模语言	即 Unified Modeling Language。是一种定义良好、功能强大的面向对象建模语言，它适用于面向对象软件开发的全过程
Uninstallation Testing	卸载测试	对软件系统可卸载性的测试
Unit Testing	单元测试	对软件中单个组件的测试
Usability Testing	可用性测试	对用户使用产品的容易程度的测试

续表

英 文 名 称	中 文 名 称	术 语 简 介
User Interface Testing	用户界面测试	目的在于测试用户界面的正确性、易用性和视觉效果
Validation	确认	目的在于判断开发的系统或组件可否追溯到用户的需求，以检验系统或组件功能及其他特性的有效性
Validation Testing	确认测试	目的在于检查已实现的软件系统是否满足了需求规格说明书中规定的各种需求，以及软件配置是否完全正确
Verification	验证	目的在于检验系统或组件是否实现了预先定义的功能及其他特性
Volume Testing	容量测试	目的在于使系统承受超额的数据容量来确定系统的容量瓶颈，进而优化系统的容量处理能力
Walkthrough	走查	是一种非正式的同行评审手段，走查小组人员设计一批有代表性的测试用例，依照程序的逻辑人工地执行它们，记录变量在运行过程中的状态，以分析程序的逻辑正确性
White-box Testing	白盒测试	在清楚组件内部逻辑结构和处理过程的前提下，检查组件内部结构是否达到了预期设计要求
XP	极限编程	即 eXtreme Programming。是以构造符合客户需要的软件为目标的方法论，使开发者能够更有效地响应客户的需求变化；突出了人在软件开发过程中的作用；属于轻量级的方法，认为文档、架构不如编程直接、有效

附录 B

优秀的测试站点资源

网　　址	简　　介
http://bdonline.sqe.com/	一个关于网站测试方面的网页
http://citeseer.nj.nec.com/	一个丰富的电子书库，且提供著作的相关文档参考和下载
http://groups.yahoo.com/group/LoadRunner	性能测试工具 LoadRunner 的一个论坛
http://rexblackconsulting.com/Pages/publications.htm	Rex Black 的个人主页，有一些测试和测试管理方面的资料可供下载
http://satc.gsfc.nasa.gov/homepage.html	软件保证中心是美国国家宇航局(NASA)投资设立的一个软件可靠性和安全性研究中心，研究包括了度量、工具、风险等各个方面
http://seg.iit.nrc.ca/English/index.html	加拿大的一个研究软件工程质量方面的组织，可以提供研究论文的下载
http://sepo.nosc.mil	内容来自美国 SAN DIEGO 的软件工程机构(Software Engineering Process Office)主页，包括软件工程知识方面的资料
http://www.asq.org/	世界上最大的质量团体组织之一，有丰富的论文资源，但是收费
http://www.benchmarkresources.com/	提供有关标杆方面的资料，也有一些其他软件测试方面的资料
http://www.betasoft.com/	包含一些流行测试工具的介绍、下载和讨论，还提供测试方面的资料
http://www.cc.gatech.edu/aristotle/	Aristotle 研究组织，研究软件系统分析、测试和维护等方面的技术，在测试方面的研究包括了回归测试、测试套最小化、面向对象软件测试等内容，该网站有丰富的论文资源可供下载
http://www.computer.org/	IEEE 的电子图书馆，有丰富的计算机方面的论文资料
http://www.cs.york.ac.uk/testsig/	约克大学的测试专业兴趣研究组网页，有比较丰富的资料下载，内容涵盖了测试的多个方面，包括测试自动化、测试数据生成、面向对象软件测试、验证确认过程等
http://www.esi.es/en/main/	ESI(欧洲软件组织)，提供包括 CMM 评估方面的各种服务
http://www.europeindia.org/cd02/index.htm	一个可靠性研究网站，有可靠性方面的一些资料提供参考

续表

网　　址	简　　介
http：//www. fortest. org. uk/	一个测试研究网站，研究包括了静态测试技术（如模型检查、理论证明）和动态测试（如测试自动化、特定缺陷的检查、测试有效性分析等）
http：//www. grove. co. uk/	一个有关软件测试和咨询机构的网站，有一些测试方面的课程和资料供下载
http：//www. hq. nasa. gov/office/codeq/relpract/prcls-23. htm	NASA 可靠性设计实践资料
http：//www. io. com/～wazmo/	Bret Pettichord 的主页，其中一个热点测试页面连接非常有价值，从中可以获得相当大的测试资料
http：//www. iso. ch/iso/en/ISOOnline. frontpage	国际标准化组织，提供包括 ISO 标准系统方面的各类参考资料
http：//www. isse. gmu. edu/faculty/ofut/classes/ 821-ootest/papers. html	提供面向对象和基于构架的测试方面著作的下载
http：//www. ivv. nasa. gov/	NASA 设立的独立验证和确认机构，该机构提出了软件开发的全面验证和确认，在此可以获得这方面的研究资料
http：//www. kaner. com/	著名的测试专家 Cem Kaner 的主页，里面有许多关于测试的专题文章
http：//www. library. cmu. edu/Research/EngineeringAndSciences/CS+ECE/index. html	卡耐基·梅隆大学网上图书馆，有大量的计算机方面论文资料
http：//www. loadtester. com/	一个性能测试方面的网站，提供有关性能测试、性能监控等方面的资源
http：//www. mtsu. edu/～storm/	软件测试在线资源，包括提供目前有哪些人在研究测试，测试工具列表链接，测试会议，测试新闻和讨论，软件测试文学（包括各种测试杂志，测试报告），各种测试研究组织等内容
http：//www. psqtcomference. com/	实用软件质量技术和实用软件测试技术国际学术会议宣传网站
http：//www. qacity. com/front. htm	测试工程师资源网站，包含各种测试技术及相关资料下载
http：//www. qaforums. com/	关于软件质量保证方面的一个论坛，需要注册
http：//www. qaiusa. com/	QAI 是一个提供质量保证方面咨询的国际著名机构，提供各种质量和测试方面的证书认证
http：//www. qualitytree. com/	一个测试咨询提供商，有一些测试可供下载，有几篇关于缺陷管理方面的文章值得参考
http：//www. rational. com/	IBM Rational 的官方网站，可在这里寻找测试方面的工具信息
http：//www. riceconsulting. com/	一个测试咨询提供商，有一些测试资料可供下载，但不多
http：//www. satisfice. com/	包含 James Bach 关于软件测试和过程方面的很多论文，尤其在启发式测试策略方面值得参考

续表

网　址	简　介
http: //www. satisfice. com/seminars. shtml	一个黑盒软件测试方面的研讨会,主要由测试专家 Cem Kanar 和 James Bach 组织
http: //www. sdmagazine. com/	软件开发杂志,经常会有一些关于测试方面好的论文资料,同时还包括了项目和过程改进方面的课题,并且定期有一些关于质量和测试方面的问题讨论
http: //www. sei. cmu. edu/	软件工程组织 SEI 的网站,在这里可免费获得各类关于软件工程质量和测试方面的资料
http: //www. soft. com/Institute/HotList/	提供了网上软件质量热点链接,包括专业团体组织链接、教育机构链接、商业咨询公司链接、质量相关技术会议链接、各类测试技术专题链接等
http: //www. softwaredioxide. com/	包括软件工程(CMM,CMMI,项目管理)、软件测试等方面的资源
http: //www. softwareqatest. com/	软件质量/测试资源中心。该中心提供了常见的有关测试方面的 FAQ 资料,各质量/测试网站介绍,各质量/测试工具介绍,各质量/策划书籍介绍,以及与测试相关的工作网站介绍
http: //www. softwaretestinginstitute. com	一个软件测试机构,提供软件质量/测试方面的调查分析,测试计划模板,测试 WWW 的技术,如何获得测试证书的指导,测试方面书籍介绍,并且提供了一个测试论坛
http: //www. sqatester. com/index. htm	一个包含各种测试和质量保证方面的技术网站,提供咨询和培训服务,并有一些测试人员的社团组织,特色内容是缺陷处理方面的技术
http: //www. stickyminds. com/	提供关于软件测试和质量保证方面的当前发展信息资料,论文等资源
http: //www. tantara. ab. ca/	软件质量方面的一个咨询网站,有过程改进方面的一些资料提供
http: //www. tcse. org/	IEEE 的一个软件工程技术委员会网站,提供技术论文下载,并有一个功能强大的分类下载搜索功能,可以搜索到测试类型、测试管理、测试分析等各方面资料
http: //www. testing. com/	测试技术专家 Brain Marick 的主页,包含了 Marick 研究的一些资料和论文,该网页提供了测试模式方面的资料
http: //www. testingcenter. com/	有一些测试方面的课程体系
http: //www. testingconferences. com/asiastar/home	著名的 AsiaStar 测试国际学术会议官方网站
http: //www. testingstuff. com/	Kerry Zallar 的个人主页,提供一些有关培训、工具、会议、论文方面的参考信息
http: //www. 51testing. com/	51Testing 软件测试网,国内的软件测试网站。介绍测试技术、测试工具、测试管理、软件质量等方面的内容,设有测试论坛
http: //www. testage. net/	测试时代,国内的软件测试网站。介绍测试技术、自动测试技术、软件质量保证等方面的内容,设有测试论坛
http: //www. uml. org. cn/	UML 软件工程组织,国内网站。全面介绍软件工程中的技术、管理理念与方法、工具等,包含了软件测试的内容

参考文献

1. 赵瑞莲. 软件测试. 北京：高等教育出版社，2004.
2. 齐治昌等. 软件工程. 北京：高等教育出版社，1997.
3. 古乐，史九林. 软件测试技术概论. 北京：清华大学出版社，2004.
4. 柳纯录等. 软件评测师教程. 北京：清华大学出版社，2005.
5. 飞思科技产品研发中心. 实用软件测试方法与应用. 北京：电子工业出版社，2003.
6. 殷人昆. 软件工程复习与考试指导. 北京：高等教育出版社，2001.
7. 朱少民等. 软件测试方法与技术. 北京：清华大学出版社，2005.
8. 贺平. 软件测试教程. 北京：电子工业出版社，2005.
9. 王健等. 软件测试员培训教材. 北京：电子工业出版社，2003.
10. Ron Patton. 软件测试. 周予滨等译. 北京：机械工业出版社，2003.
11. 张雪萍. 面向对象软件类测试研究新进展. 计算机工程与设计，2006，27(11)：1954～1956.
12. 龚红仿等. 面向对象的软件测试技术探讨. 长沙理工大学学报(自然科学版)，2004，1(2)：68～71.
13. 杨善红，李静雯. 测试驱动开发研究. 黑龙江科技信息，2007，5：52～53.
14. 中华人民共和国国家标准. 计算机软件测试文件编制规范 GB/T 9386-1988. 北京：中国标准出版社：1989.
15. Rational Software Corporation. Rational Unified Process，2000-02-03.
16. http：//blog. csdn. net/ccsdba/archive/2006/06/22/822167. aspx.
17. http：//www. itisedu. com/phrase/200604231331545. html.
18. http：//www. sqstudy. org/Models/Testing_Quality/Choose_Automation_Testing_Solution. htm.
19. http：//blog. csdn. net/chenshaoying/archive/2006/07/03/869867. aspx.
20. http：//www. etesting. com. cn/html/c/c2/124_1. html.
21. http：//www. 51testing. com/html/69/1301. html.
22. http：//www. rapidtesting. cn/Html/TestStyle/061532. html.
23. http：//www. ibm. com/developerworks/cn/rational/r-test/index. html.
24. http：//blog. csdn. net/VSaber/archive/2002/02/08/8545. aspx.
25. http：//www. 51testing. com/？ action_viewnews_itemid_6550. html.
26. http：//www. ibm. com/developerworks/cn/linux/l-tdd/index. html.
27. http：//www. itisedu. com/phrase/200603121128285. html.